SOFTWARE ENGINEERING LABORATORY SEL-90-004

Gamma Ray Observatory Dynamics Simulator in Ada (GRODY)

Experiment Summary

SEPTEMBER 1990

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

SOFTWARE ENGINEERING LABORATORY

SEL-90-004

Gamma Ray Observatory Dynamics Simulator in Ada (GRODY)

Experiment Summary

SEPTEMBER 1990

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC, Systems Development Branch

The University of Maryland, Computer Sciences Department

Computer Sciences Corporation, Systems Development Operation

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document.

The major contributors to this document are

T. McDermott (CSC)

M. Stark (GSFC)

Single copies of this document can be obtained by writing to

Systems Development Branch
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

This document is the final report on the Gamma Ray Observatory (GRO) Dynamics Simulator in Ada (GRODY) experiment. This experiment involved the parallel development of dynamics simulators for GRO in both FORTRAN and Ada for the purpose of evaluating the applicability of Ada to the National Aeronautics and Space Administration/Goddard Space Flight Center's (NASA/GSFC's) flight dynamics environment.

GRODY successfully demonstrated that Ada is a viable, valuable technology for use in this environment. In addition to building a simulator, the Ada team evaluated training approaches, developed an Ada methodology appropriate to the flight dynamics environment, and established a baseline for evaluating future Ada projects.

Table of Contents

PRECEDING PAGE BLANK NOT FILMED

PAGE VI INTENTIONALLY BLANK

List of Illustrations

List of Tables

EXECUTIVE SUMMARY

INTRODUCTION

The Software Engineering Laboratory (SEL) performed a study of Ada technology by conducting a parallel development effort for a flight dynamics simulator, a type of system that is typically composed of approximately 50,000 source lines of code (SLOC). The experiment was initiated in January 1985 with final analysis being completed in June 1990.

OBJECTIVE AND SCOPE

The objective of the study was to assess the impact of Ada technology in a production environment. The study involved two separate development teams, one using Ada and the other FORTRAN, building simulators from the same specification. The scope of the study is a comparison of these projects.

RESULTS

The results of the study are based on both objective and subjective measures of the two projects. The primary results are listed as follows:

- **Training** for Ada is most effective when it ensures that developers understand the software engineering principles embodied in Ada, the design methodology to be used, Ada syntax and semantics, and any vendor-specific features of the Ada environment, such as input/output details or the library management system. Managers and reviewers also need training.
- **Effort distribution among life cycle phases** was nearly unchanged.
- **Productivity** measured as code development rate was slightly higher in Ada. The Ada system consumed more effort since it was larger. GRODY's extensive new technology development impacted both productivity and total effort.
- **Reliability** was lower with Ada but was also primarily an effect of the first use of Ada.
- **Design** characteristics were very different with Ada. The Ada design directly reflected software engineering principles, such as hierarchical structure and information hiding.
- **Code** required more source lines with Ada but was more readable. Counting SLOC, the Ada system was 2.5 times larger than the FORTRAN system; counting statements, it was 1.5 times larger.

- **Testing** showed little difference between the two languages, which is to be expected since the SEL performs functional testing that reduces the impact of the implementation language.
- **Team satisfaction** was higher with Ada. At the end of the project, the Ada team requested assignment to Ada projects, and a number of the FORTRAN developers also switched to Ada.
- **The General Object-Oriented Design (GOOD) Methodology** was developed to meet the specific needs of the flight dynamics environment.

SECTION 1—INTRODUCTION

This document is the summary and final report on the Gamma Ray Observatory (GRO) software development experiment. This experiment was conducted by the Flight Dynamics Division's (FDD's) Software Engineering Laboratory (SEL) and the Data Systems Technology Division. The experiment evaluated the suitability of Ada for use in the flight dynamics area by developing, in parallel, two attitude dynamics simulators for the GRO satellite, one written in FORTRAN (GROSS) and one in Ada (GRODY).

This document collects and summarizes previously published results, with adjustments in those cases where preliminary analyses were based on interim data that differ from the final data. Appendix B lists the published articles and documents that resulted from the study.

1.1 DOCUMENT STRUCTURE

This section provides the introduction and background for the experiment. Section 2 presents the experiment's project plan, and Section 3 describes the experiment and presents results.

1.2 EXPERIMENT BACKGROUND

This section describes the flight dynamics environment in which the SEL conducts its experiments and the type of application developed during the GRODY experiment.

1.2.1 The Flight Dynamics Environment

Software developed in the flight dynamics area is typically ground-based, non-embedded, scientific (algorithmic) in nature. These applications include spacecraft attitude determination, attitude control, maneuver planning, orbit determination, orbit correction, and mission analysis. Systems usually comprise between 30,000 and 200,000 source lines of code (SLOC). Between 15 and 30 percent of the total source is typically reused from other projects.

The commonly used language has been FORTRAN. The SEL models of software development, such as effort distribution by phase, reflect the history of using FORTRAN. System specifications have implicitly assumed the traditional function-oriented, structured, top-down decomposition approach to software development.

1.2.2 Dynamics Simulators

Figure 1-1 illustrates the operation of a dynamics simulator. For each cycle of the simulation, the onboard computer (OBC) model (on the right side of the figure)

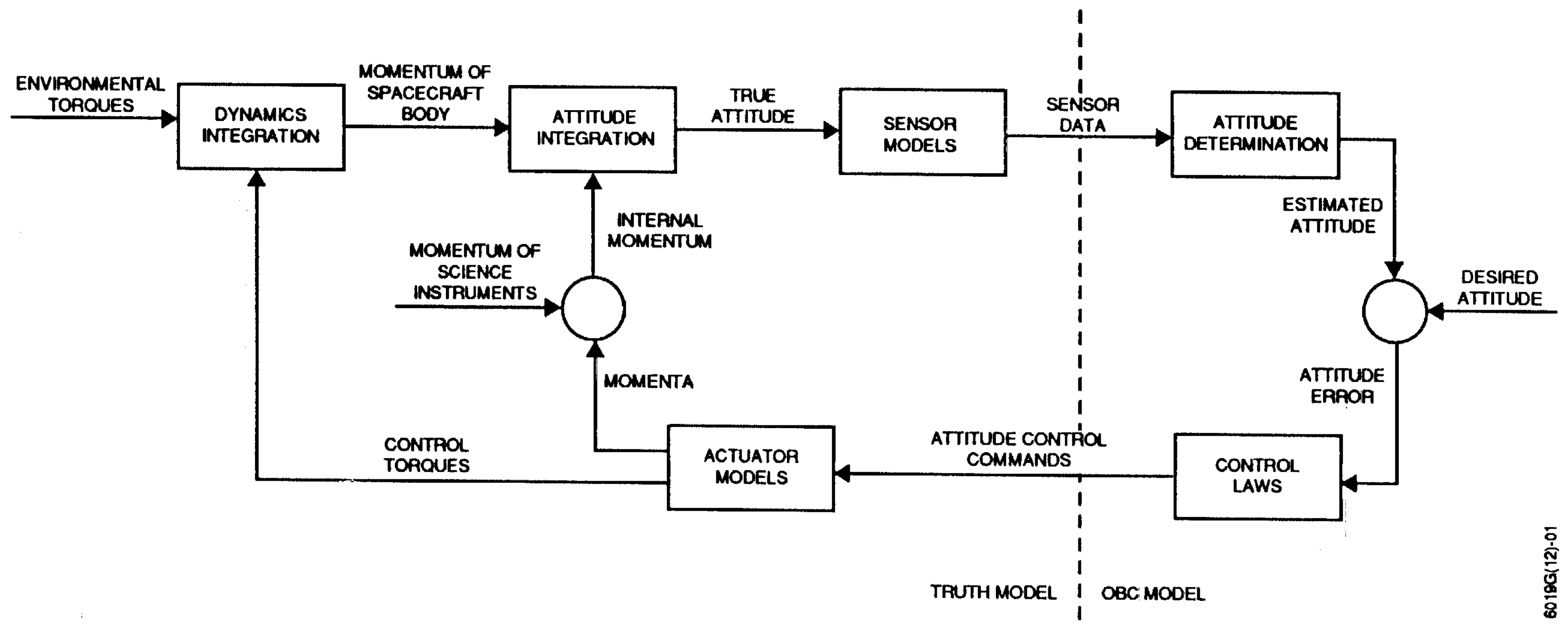

Figure 1-1. Operation of a Dynamics Simulator

uses sensor data to compute an estimated attitude. The estimated attitude is compared with the desired attitude to calculate the attitude error. The OBC applies control laws to generate commands for the attitude actuators to correct the attitude error. The simulator truth model accepts the actuator commands, calculates the actuator and spacecraft responses, and produces the true attitude of the spacecraft. The truth model then uses the true attitude to produce sensor data that are sent to the OBC. This completes one simulation cycle.

1.2.3 The Software Engineering Laboratory

The SEL was founded in 1977 as a cooperative effort of the National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC), University of Maryland, and Computer Sciences Corporation (CSC). The goals of the SEL are to understand software development in the flight dynamics environment, to evaluate new software technology, and to insert appropriate technology into the flight dynamics software development process.

The SEL has, over time, refined a set of data collection forms (*Data Collection Procedures for the Rehosted SEL Data Base*, SEL-87-008) tailored to monitoring software development in the flight dynamics area. Special data collection procedures for the GRODY study are discussed in Section 2.3.3. One purpose of the GRODY experiment was to initiate a baseline for evaluation of systems developed in Ada.

SECTION 2—PROJECT PLAN

The overall goal of the experiment was to develop insight into the applicability of the Ada development methodology and language in the NASA software environment. The project plan identified six objectives as means of attaining the goal along with sets of questions pertaining to each objective. Section 2.1 discusses the objectives, and Section 2.2 lists the questions. The project plan is included in this report as Appendix A.

The project plan should be considered as a charter, establishing which issues were to be considered for study, rather than a set of minimum requirements. The experimenters were to refine and focus the investigation as they gathered information and developed understanding.

2.1 EXPERIMENT OBJECTIVES

The objectives were to determine the following six items of interest:

- Cost effectiveness of adopting Ada in specified environments
- Set of software measures useful in determining the advisability of adopting Ada for portions of the Space Station project
- Trade-offs and characteristics of the potential impact of using Ada as a development methodology and language
- Impact of Ada on the reuse of software
- Profiles (characteristics) of the development process using Ada as opposed to using classical high-order languages
- Effect of Ada (methodology and language) on productivity, reliability, maintainability, reusability, and portability

The two highest priority objectives of the study were to understand the Ada development process and to establish and evaluate baseline measurements for Ada development. For the purposes of this report, the objective of determining the cost effectiveness of Ada will be treated as a subset of determining the effect of Ada.

2.2 QUESTIONS TO BE ANSWERED

For purposes of exposition, this report has arranged the questions from the experiment plan into questions with narrative answers and questions with quantitative answers. The narrative questions are grouped under the objective of characterizing the Ada development process; the quantitative questions are grouped under the objective of determining the effects of Ada methodology and language.

2.2.1 Characterization of the Ada Development Process

Understanding the Ada development process was more difficult than envisioned in the project plan. The plan reasonably expected, based on the Ada literature and vendor offerings, that Ada methodology was sufficiently defined and available to allow the experiment to import appropriate approaches to Ada development. One effect of this expectation is the heavy concentration of questions concerning training.

Fortunately, the experimenters were allowed the flexibility to adjust the study. The plan is nearly devoid of any questions concerning the proper use of Ada, although some of the most valuable contributions were in the development of Ada methodology.

2.2.1.1 TRAINING

The training questions were as follows:

- How expensive was the Ada training, and could training be more cost effective?
- How can Ada (versus other languages) training requirements be determined?
- Are people with certain types of academic backgrounds and professional experiences more readily adaptable to Ada?
- Was the training sufficient or was it supplemented as problems developed?
- Can a sufficient training program be developed from this experience?
- How should training be improved?

2.2.1.2 THE ADA PROCESS

Three software development process questions were posed:

- Did the Ada project meet its deadlines?
- Can the impact of 'deleting' software heritage in specific environments be assessed?
- Is the completed Ada code more readable and easier to comprehend than other languages?

2.2.2 Effect of Ada Methodology and Language

The following questions are quantitative:

- What is the relative manpower cost of training, designing, coding, testing, and changing Ada software versus FORTRAN software?
- What was the relative cost of each phase?

- Excluding training, how did productivity on the two projects differ?
- What was the change rate of software during the development (specifically, change and errors due to unfamiliarity with Ada)?
- What was the relative effort required for making a change? Fix an error?
- How did the failures in the two developments differ in type and severity?
- What is the relative effort required to implement a change in requirements?

2.3 PROCEDURES

The project plan included procedures for conducting the study. The following sections discuss the organization of the experiment, data collection procedures, and computer resources, with key quotations from the project plan.

2.3.1 Team Structure

The project plan defined the team structure for the experiment as follows:

> Supporting this Ada experiment will be personnel from three distinct installations—NASA/GSFC, CSC, and the University of Maryland. There will also be three functional teams or groups that will be part of this project; the Ada development team, the FORTRAN development team, and the study group directing the experiments.

2.3.1.1 ADA DEVELOPMENT TEAM

> The team responsible for the design, implementation and test of the dynamics simulator utilizing the Ada development methodology and language will consist of approximately seven software developers. Each of these people will be devoting a minimum of 1/3 time with an average of 1/2 time being allocated by each person. These resources are expected to be provided from the following organizations. (It is estimated that the project will continue at least through July of 1986.)

Throughout Ada development (including the training phase), 7.9 people per week charged time to GRODY. The average time charged was 14.9 hours.

2.3.1.2 FORTRAN DEVELOPMENT TEAM

> The team responsible for the implementation of the FORTRAN version of the dynamics simulator will be comprised of personnel from CSC and from Code 550. The project will be handled exactly as any other

development effort under the PC&A contract. The implementation team will be comprised of the following:

Code 550: 3 developers, each allocating from 1/2 to full time on the project.

CSC: 3 to 4 developers, each allocating from 1/2 time to full time.

It is not anticipated that any special interface or contact will be required because of the nature of the experiment on the FORTRAN team. There needs to be no interface with either the University of Maryland or with Code 520. It is expected that SEL data will be provided, as usual, by the FORTRAN team.

An average of 7.2 people per week charged to GROSS. The average time charged was 16.6 hours per week. The GROSS and GRODY effort averages do not match the subjective impressions of the nature of the two projects. GROSS was a production product, with people assigned essentially full time. GRODY was characterized by periods of intense activity separated by fallow periods.

2.3.1.3 ADA EXPERIMENT STUDY GROUP

The team responsible for defining, directing and analyzing the overall Ada project experiment is to consist of staffing from GSFC, the University of Maryland and CSC. This team will be responsible for defining the overall experiment (including measures, goals, training required, data to be collected) as well as to analyze results. Specifically, the team will

- Develop plans, goals, objectives and data to be collected
- Define the procedures for carrying out the project
- Develop and write the overall project plan
- Develop and provide (or make available) all required training and project preparations
- Monitor project development and recommend any required adjustments (such as staffing, resources, training)
- Complete analysis of the project and develop report(s) describing these results.

2.3.2 Team Interaction

There will be a close working relationship between the Ada development team and the Ada Experimenter team. This will include periodic meetings to review status, difficulties, etc. in an attempt to identify additional training needed or to identify and help in certain problems with the Ada language or methodology.

Although there will be no attempt to completely isolate the FORTRAN team from the Ada development team, there will be no attempt to encourage interaction. Formal procedures will be defined whereby requirements change, errors, and general modifications will be made known to both teams in the same timeframe and format. No attempt will be made to provide questions/answers generated by one team to the other team. The designs, development methodologies, and general procedures of one team will not be reviewed by the other team.

The only interaction between the FORTRAN team and the Ada Experimenters will be for the experimenters to periodically review the data and information being recorded by the FORTRAN team.

2.3.1.4 COMPUTER SUPPORT

The project plan also defined the required computer support. It should be noted that all FORTRAN development took place on a VAX 11/780 and that the Ada project migrated to a VAX 8600 during implementation.

In order to support the development efforts of the Ada project, computer support required will be supplied in the following manner:

- VAX 11/780 Development Machine - time, terminals, and necessary access will be supplied by the Code 520 VAX and by the Code 550 VAX. All team members are to have access to either machine on a continuing basis. Accounts are to be provided to all team members and managers on both computers. For the processors not normally scheduled to be in operation on off-hours, special requests may be made to access the processor at nights and/or weekends.
- Compilers/tools - Both VAX machines will house the same Ada compiler and support environment.

2.3.3 Special Data Collection

The project plan included specific data collection procedures for the Ada project:

Programmer/Analyst Survey - All programmers, managers and librarians will complete this form as soon as they begin work on the project. This includes the Ada team and the FORTRAN team.

Resource Summary Form - This form will be completed on a weekly basis, beginning on January 1, 1985, by all team members and managers. Total weekly hours expended on the Ada project are recorded.

Computer resources will be provided by the librarian through the VAX accounting.

Component Status Report - This form will be completed on a weekly basis beginning January 1, 1985, by all team members. Personnel devoting 100 percent of their project time to "management" will not complete this form.

Component Origination Form - This form will be completed whenever a particular "component" of the project is identified (design). An Ada component is defined as a task, package or subprogram. The person first identifying the component is responsible for completing the form.

Change Report Form (1 and 2) - Both the standard SEL change report form, as well as the added page of questions will be completed by the Ada team. The forms are to be completed for any change/error defined after a programmer has completed the component origination form. The form will be completed by the individual making the change.

Project Estimation Form - This form will be completed by the project technical manager each 6 weeks beginning in early January. The form may be completed in consultation with other team members.

Project Header File Information Form - This form contains all of the actual sizes and characteristics of the completed project. It will be completed <u>once</u> at project completion by the technical manager.

In addition to the forms to be utilized to collect data, two additional sets of information will be saved:

- <u>Change/Growth History</u> - Each week of the project, the support librarian will record the number of lines of code, number of components and total number of changes that have been made to the source library.

- <u>Subjective File Data</u> - At the completion of the project, a set of parameters which characterizes the development process and product will be determined by the managers of the development effort.

SECTION 3—PROJECT RESULTS

3.1 CHARACTERIZATION OF THE ADA DEVELOPMENT PROCESS

This section presents responses to the narrative study questions. Training is considered first, followed by a discussion of activities in each of the life cycle phases.

3.1.1 Training

From the outset of the experiment, training was known to be central to successful adoption of Ada technology. The project plan states "... training of the development team is both very critical and very difficult. Responding to experiences of other projects attempting to utilize Ada, an extensive effort is to be put forth in the training of the Ada Team."

The training program included instruction in both the Ada language and design methodologies. The training program consisted of four steps:

1. Reading *Software Engineering with Ada* by Grady Booch.
2. Viewing a set of videotaped tutorials from Alsys, Inc. This took 40 hours of classroom time, with class discussion following each tape.
3. Participating in a 3-day seminar on the Process Abstraction Method design methodology by George Cherry of Language Automation Associates.
4. Implementing a practice problem using the DEC Ada compiler. This system was an electronic mail system comprising 5700 SLOC, 1400 of which were executable. The practice problem took 1336 hours of effort.

The GRODY team found discussions in class, team meetings, and the practice problem to be the most useful parts of the training. Unfortunately, the practice problem was larger than it needed to be. The team felt a smaller exercise that preserved the need to use packages and data abstraction would have been more effective training.

The GRODY team found the following aspects of Ada to be either especially difficult or insufficiently covered by the training:

- Input/Output (I/O)—Every training resource the team used was sketchy on I/O operations. This is because efficient I/O is machine-dependent and cannot be covered in general training.
- Tasking—Tasking was covered by all of the training resources, but there are specifics that the language definition leaves open to compiler writers'

interpretation. An example is how a "select" statement chooses among multiple conditions. Some compilers always handle the first condition in the code, while others make an attempt at fairness.

- Generics—Generics are theoretically straightforward but caused problems during the practice problem.

- Data Types—This area needs special emphasis, especially for people without a computer science background, since it is a primary mechanism for information hiding. If the software engineering concepts behind data types are not understood, not all the benefits of Ada will be fully realized.

- Library Units and Library Structures—The concepts of separate compilation and program libraries were covered by the training but needed to be augmented with specifics of the compiler vendor's library structure.

The GRODY study concluded that Ada training should cover not only the language but also the related software engineering concepts, the chosen design methodology, and, ideally, several other methods as well. Managers and reviewers should also be trained in the design methodology to be used so they have a proper basis for evaluating a project's progress. Specifically, the study identified several levels of training needed for an effective Ada team:

- Ada syntax—How to construct code that the compiler will accept.

- Ada semantics—How to construct programs that do what is intended.

- Design principles—How to analyze problems and synthesize solutions that take advantage of Ada's features.

- General methodologies—The more views a designer can apply to a problem, the more likely he or she is to produce a good design.

- Project methodology—Each team member must be intimately familiar with the specific methodology that is selected for use on the project.

The answers to the training questions are given below.

Q. **How expensive was the Ada training, and could training be more cost effective? Can a sufficient training program be developed from this experience?**

A. Counting the practice problem, training consumed 2436 staff hours, or nearly 11 percent of the total hours charged for the GRODY development. GRODY tried multiple training approaches; a focused training program drawn from the lessons learned on this experiment would almost certainly be more cost-effective and efficient for flight dynamics.

Q. **Are people with certain types of academic backgrounds and professional experiences more readily adaptable to Ada?**

A. Table 3-1 lists the experience attributes of the two teams. The GRODY team had more general software experience than the GROSS team but had less experience with dynamics simulators. The Ada team knew twice as many computer languages as the FORTRAN team, but polyglot programmers proved to have no advantage in learning Ada. The Ada team reported that prior experience with the specific concepts of exception handling and concurrency would have been beneficial. The answer to the question is that there are no obvious ways to predict an individual's success with Ada.

Table 3-1. Team Experience

Experience Factor	GRODY	GROSS
Years of software engineering experience	8.6	4.8
Median number of applications experience	4	3
Mean number of computer languages	7.0	3.0
Fraction of team with dynamics simulator experience	43%	66%

6019

Q. **Was the training sufficient or was it supplemented as problems developed?**

A. The language-specific training was adequate. The methodological training was sufficient for the methodologies taught, but those methodologies were not sufficient for developing dynamics simulators. The team had to develop their own methodology.

Q. **How should training be improved?**

A. Training can be improved by incorporating GRODY's lessons learned, such as including specific flight dynamics insights, in the design training. Vendor-specific training should also be included.

Q. **How can Ada (versus other languages) training requirements be determined?**

A. This question was not answered in the study.

3.1.2 Development Process

The greatest contribution of the GRODY experiment was the development of an Ada methodology suited to the flight dynamics environment. The theme of the following subsections is the search for, discovery, and refinement of that methodology and its influence on the project life cycle phases.

Only one narrative question was asked about the development process:

Q. **Did the Ada project meet its deadlines?**

A. No, this was, however, an experimental artifact rather than an effect of using Ada. The original schedule assumed that this experiment would be an evaluation of existing Ada technology. As the project progressed, it became clear that the Ada team would have to develop its own methods. Meeting the schedule became less important. Figure 3-1 shows the original and final schedules of GROSS and GRODY.

3.1.2.1 REQUIREMENTS ANALYSIS

The specification and requirements documents provided the functional requirements of the simulator. They also contained a high-level design that was used for the FORTRAN development. The documents are organized into subsystems that correspond to the structure of the last several successful dynamics simulators. The FORTRAN heritage was so strong that the experiment plan even contained an abbreviated version of the design.

One goal of the study was to investigate Ada design methods. It was clear that the GRO dynamics simulator specification incorporated too much FORTRAN legacy to permit the GRODY team to explore design options. Consequently, the team took the time to recast the specification in a more language-neutral form. The team used a specification approach called the Composite Specification Model (CSM) (*Guidelines for Applying the Composite Specification Model (CSM)*, SEL-87-003). CSM represents a system's requirements with functional, dynamic, and contextual views. The GRODY team had no trouble using CSM and felt that the new specification represented the system requirements without imposing a language bias. The specification exercise also allowed the team to become more familiar with the system it was trying to develop. This was beneficial, since the team had minimal experience developing dynamics simulators.

The GRODY team produced two documents from this requirements analysis phase, a rewritten requirements specification and a requirements assessment report.

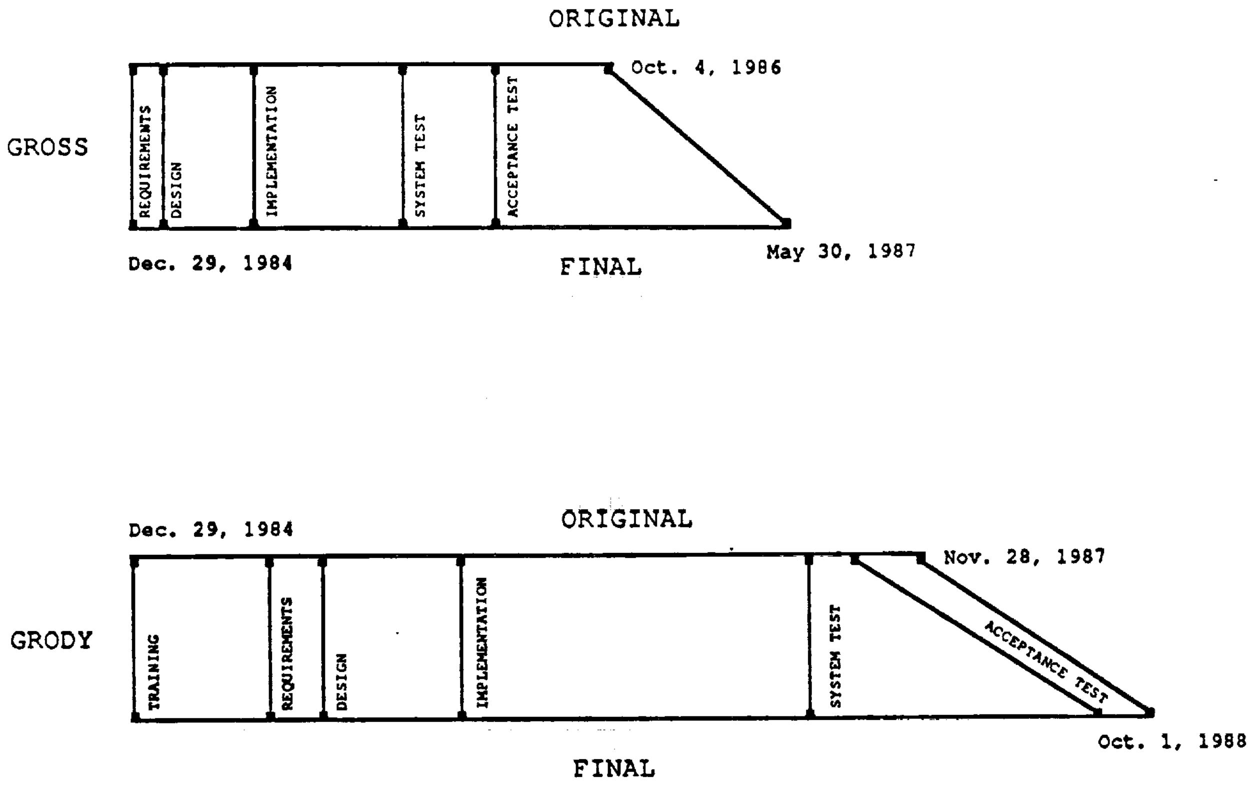

Figure 3-1. Original and Final Schedules

Only one question was germane to the requirements analysis phase:

Q. **Can the impact of "deleting" software heritage in specific environments be assessed?**

A. No, the effort data presented in Section 3.2 are inconclusive. There are costs associated with experimenting with any new design or technology. GRODY used both a new technology and a new design, but the costs were not explicitly counted. Perhaps a better way to phrase the question is to try to assess the savings that flow from having a long history of using one technology and design.

3.1.2.2 DESIGN

To try to fully exploit the Ada language, the GRODY team wanted to use a design methodology that was well-suited to Ada and its particular features, such as packages, information hiding, and tasks. The team explored, to the point of developing high-level designs, three different methodologies.

The first method used was developed by Grady Booch. It calls for translating a textual specification into a design by underlining nouns and verbs in the specification. The nouns map into objects, and the verbs map into operations. Booch defines a graphic notation that includes symbols for packages, tasks, visible data types, visible procedures, functions (or entities), and hidden code.

The second methodology investigated was the Process Abstraction Method for Embedded Large Applications (PAMELA). PAMELA was developed by George Cherry for use in real-time and embedded systems. PAMELA provides symbols to represent primitive and nonprimitive processes, rendezvous between processes, data flow, and limited control flow.

The third methodology, General Object-Oriented Design (GOOD) (*General Object-Oriented Development*, SEL-86-002), was developed by the team itself, beginning during project training and continuing throughout design. This method tries to combine the best features of the other two methods while expanding and generalizing the techniques. The methodology's notation, called object diagrams, provides symbols for objects, control flow between objects, and the decomposition of objects into lower level objects. Object diagrams illustrate the control structure of the system, and the object descriptions define the data flow in the various operations. Object diagrams are drawn with the senior-level controlling objects at the top of the page and the junior-level controlled objects below them. This serves as a graphic mechanism for demonstrating system hierarchy.

The GRODY team selected the GOOD methodology because it best fit its needs. The object diagram methodology was used to complete preliminary design and detailed design.

The GRODY team had some difficulty with life cycle phase boundaries, since the SEL guidelines are tailored to FORTRAN developments. The GRODY design reviews were schedule-driven, rather than occurring when the design was ready for review.

The GRODY team presented its preliminary design review (PDR) 1 month into design, using GOOD notation. The team presented its critical design review (CDR) 4 months later. Not all package specifications had been written at CDR, obviating the benefits of having the compiler check static interfaces during design.

Part of this delay was a duplication of effort needed to translate "object descriptions" into Ada package specifications. This was essentially mechanical transcription and could have been avoided by using Ada itself as the design notation.

The method of building object diagrams was not as explicit as either the Booch approach or PAMELA, but it was guided by the principles of information hiding, abstraction, and design hierarchy. A few problems were noted with the design. The strong coupling between some of the objects was not effectively shown by the early design notation. The notation was modified to show these relations. In fact, the design methodology evolved throughout the design period.

The following conclusions were drawn from the design phase of this study:

- Training is important. Ada training should cover not only the language but also the related software engineering concepts, the chosen design methodology, and, ideally, several other methods as well. Managers and reviewers should have some training in the design methodology to be used.
- Specification should not constrain design.
- An appropriate methodology is important. The methodology chosen for a project should fit the problem. PAMELA, with its emphasis on concurrency, was not appropriate for GRODY.
- Ada documentation is different than FORTRAN documentation.
- Designing with Ada may imply different phase boundaries. At a minimum, the definition of phases and the products expected from each phase should be clear.
- Changing to a new technology imposes additional costs.
- Package specifications should be used in place of object descriptions.

3.1.2.3 IMPLEMENTATION

The GRODY implementation phase began with completion of the design work, specifically writing package specifications and program design language (PDL). The package specifications were compiled to find any static interface errors.

The GRODY design called for a global types package that defined types used throughout the system. Design of the types package was not complete prior to the start of implementation, so that the definition of new types or modification of existing types as implementation progressed forced the recompilation of large parts of the system. In retrospect, design should incorporate an abstract data type analysis to control the proliferation of data types. Starting from a small set of general types, families of subtypes could be generated, allowing the reuse of properties of the general type.

The GRODY design led to a system that, once implemented, was highly interconnected. It appears that while object-oriented design allows partitioning into independent subsystems, using object-oriented techniques does not automatically lead to good partitioning.

Another design problem that emerged during implementation was that the functionality allocated to procedures was not well communicated. During design, functions were allocated to packages, but not explicitly to procedures. Prologs described the function of procedures but did not describe the algorithm to be implemented.

The design was also vague on tasking issues. Control of rendezvous interactions was not made explicit nor were conditions for task termination. Together with the team's general lack of experience with tasking, these ambiguities made the implementation of tasks difficult.

The GRODY team decided to use nesting, rather than library units, as the primary structuring mechanism of its code. A library unit is defined as the outermost specification in a file for a package or procedure. Nesting is the encapsulation of specifications of a package, subprogram, or task inside the body of another package, subprogram, or task. While nesting seemed to be the most natural expression of the design, it caused problems during development and testing. GRODY library units were used for the top three or four levels of the design, while nesting was used another 8 to 10 levels below that.

The primary disadvantage of nesting during implementation was that it increased the amount of recompilation required. The compiler assumed compilation dependencies between sibling nested entities, whether the dependency existed or not. Nesting complicated testing since it was not possible to execute the encapsulated objects independently. Furthermore, it was not easy to identify the caller of nested modules. It is now believed that extensive use of nesting instead of library units will make maintenance more difficult.

The GRODY experience highlights the need to develop package specifications and PDL during design phase. This would require lengthening the design phase and delaying CDR. The separation of package specifications from bodies was considered beneficial, and the GRODY team recommended that specifications should be under configuration control at the start of implementation.

The following lessons were learned during the implementation phase of the study:

- Design should be completed to the level of package specifications and PDL before starting implementation, even if the design phase has to be lengthened.

- Library units, as opposed to nesting, should be used as the primary code structuring mechanism.

- If tasks are used, the design must make rendezvous interactions and task termination conditions explicit.
- The use of object-oriented design does not, in itself, ensure good partitioning of the system into subsystems.

Q. **Is the completed Ada code more readable and easier to comprehend than other languages?**

A. Yes, team members reported that Ada was much easier to read.

3.1.2.4 SYSTEM TEST

System testing included completing the integration of modules developed during implementation. The system test plan for GRODY was based on the system test plan for GROSS. This seems reasonable, since system testing should be based on the functional specification, not on the details of the implementation. There was no need to consider Ada or object-oriented design in the test plan.

In general, little difference occurred in conducting system tests between the FORTRAN and Ada systems, since neither the execution of tests nor the interpretation of results required knowledge of the system's internals. However, three Ada features required special attention.

The first feature was exception handling. The team had difficulty inducing conditions that would cause some of the system's exceptions to be raised. Although some exceptions were difficult to test, the team felt that exceptions were useful in error handling.

The second feature was the coordination of concurrent tasks for testing. Testing concurrency requires keeping track of the state of multiple parallel operations that are not necessarily synchronous. Concurrency is a challenge to test in any language; but Ada, unlike FORTRAN, has support for concurrency. Therefore, in some sense, this is an Ada issue.

Finally, the Ada rename feature occasionally caused confusion during debugging. This was more a failure of the debugger than a deficiency of the language. DEC agreed that the debugger should be fixed.

The following lessons were learned from system testing:

- Preparation and execution of system tests was not affected by the programming language.
- A good repertoire of tools is important. The symbolic debugger was invaluable.
- Ada may reduce some types of errors. Team members consistently reported that the compiler detected many of their interface errors. Objective data neither confirm nor contradict this assertion.

- Ada may be easier to debug. Team members reported Ada's readability made finding errors easier. Objective data neither confirm nor contradict this assertion.
- Recompilation of Ada units can have a significant cost.

3.2 EFFECT OF ADA METHODOLOGY AND LANGUAGE

This section presents the quantitative results of the GRODY study. These numerical results are included for completeness, but they are peripheral to the GRODY project's major contributions to advancing software engineering at NASA in particular and in the broader software community in general. GRODY's contribution was the development and dissemination of practical techniques for building Ada software and showing how ideas that had been largely theoretical could be applied to real problems.

The statistics and graphs in this section are presented in answer to specific questions in the project plan. At first glance, they seem to indicate that Ada would make a poor candidate for the language of choice for flight dynamics software developments. This conclusion would be valid only if the two projects were doing the same thing and had equally well-controlled processes. Neither of these conditions were true for this study.

The two teams did different things. The GROSS team was building an operational product. The GRODY team was developing software technology. The experiment plan expected that appropriate Ada technology was available and that the experiment could be a side-by-side comparison. Ada technology was not sufficiently mature, and the GRODY team had to invent it. As a result, the information presented here does not so much reflect the relative merits of the two languages as it contrasts product development with technology development.

The second point follows from the first. The GROSS task (product development) was well understood with a defined set of measures, warning indicators, and corrective actions. The GRODY task (technology invention and insertion) was, by definition, exploratory and ad hoc. Drawing conclusions from such different processes is tenuous.

For additional perspective on the weight that should be put on the GRODY numbers, consider the following quote from Dr. Jerry Page from a presentation to the Sixth (1981) Software Engineering Workshop about the SEL's evaluation of another technology new to the SEL environment:

> To qualify this, our experience with many methodologies has been as follows:
>
> - The first time a methodology is applied, mistakes are made (and we made mistakes), and many of the potential benefits or advantages of the methodology are not realized.

- The second time a methodology is applied, there is a tendency to overcompensate for the things that you did worst the first time, and the methodology still does not work as well as it potentially could.
- The third time a methodology is applied, you lower your expectations somewhat or modify them, and you home in on what is right for your environment.

The relevant questions from the project plan are addressed as follows:

Q. **What is the relative manpower cost of training, designing, coding, testing, and changing Ada software versus FORTRAN software?**

A. This question asks the amount of time charged to each activity, regardless of the phase in which the activity occurred. Table 3-2 presents the staff-hours recorded in the SEL data base for each activity. The "Other" category covers the hours for attending meetings, writing documentation, and other necessary but hard to categorize activities.

Table 3-2. Staff Hours by Activity

Activity	Staff-Hours (Percent)			
	FORTRAN		Ada	
Training	N/A	N/A	2436	10.6%
Requirements analysis	1311	8.6%	499	2.2%
Design	2224	14.7%	5679	24.6%
Implementation	4253	28.0%	6645	28.8%
System test	1562	10.3%	2724	11.9%
Acceptance test	1053	6.9%	23	0.1%
Other	4762	31.4%	5020	21.8%
Total	15165		23026	

6019

This table contains several interesting points. GRODY bears out the expectation that Ada projects will devote more time to up-front activities (requirements analysis and design) than will FORTRAN projects, but only if these two activities are combined. The hours for GRODY's requirements analysis seem unrealistically low, given that GRODY recast the specification into CSM while GROSS was relying on a history of experience and the FORTRAN heritage of flight dynamics. It seems likely that the effort spent on CSM was recorded under a different activity category, probably training. The GRODY training phase consumed almost 11 percent of the effort at project inception, and this further clouds evaluating the distribution of effort on remaining project activities. Implementation effort is comparable between GROSS and GRODY. The combined testing phases differ by only 5 percent, but this result is clouded by the lack of GRODY acceptance testing.

The overall similarities between the statistics imply that at least on early Ada projects, effort distribution by activity will not differ markedly from that on FORTRAN projects.

Q. **What was the relative cost of each of the phases?**

A. Table 3-3 lists the technical and line management hours charged during each phase of the two projects.

Table 3-3. Staff-Hours by Phase

Phase	Staff-Hours (Percent)			
	FORTRAN		Ada	
Training	N/A	N/A	3346	14.5%
Requirements analysis	849	5.6%	540	2.3%
Design	2830	18.7%	2987	13.0%
Implementation	5397	35.6%	11174	48.5%
System test	2315	15.3%	4968	21.6%
Acceptance test	3774	24.9%	11	0.0%
Total	15165		23026	

6019

Table 3-3 shows one important datum. GROSS spent one-quarter of the project's effort in acceptance test phase. This was an effect of the Challenger Space Shuttle disaster. Challenger was lost a few months before GROSS entered acceptance testing, which caused the grounding of the shuttle fleet and an indefinite postponement of the GRO launch. GROSS no longer faced operational deadlines. The acceptance test phase doubled from 6 months to 1 year, and GROSS implemented a large number of specification modifications and enhancements. GRODY implemented some, but not all, of these changes.

Q. **Excluding training, how did the productivity on the two projects differ?**

A. Productivity is the amount of product developed per unit of effort. Therefore, to answer the question, the sizes of the two systems and the effort expended in building them must be determined. Table 3-4 presents the relative sizes of

Table 3-4. Size Measurements

Measure	GRODY	GROSS	Ratio
SLOC	128261	51704	2.5 : 1
Statements	40561	27642	1.5 : 1
Developed LOC	125715	39692	3.2 : 1
Developed statements	39868	22695	1.8 : 1
Comments	20700	22409	

6019

the two systems. SLOC is, in the SEL, a count of all physical lines in the program. Statements is a count of complete instructions; an assignment statement continued over three physical lines counts as one statement. Developed SLOC and statements are measures that account for savings from reuse. Developed SLOC (or statements) are computed by counting all new and extensively modified code and adding 20 percent of the unchanged and slightly modified code. GRODY had 2 percent reuse; GROSS had 29 percent reuse.

Table 3-4 shows that a system implemented in Ada is bigger than a corresponding FORTRAN system. Counting statements, which should eliminate most coding style differences, GRODY was 50 percent larger than GROSS. An unexpected observation was discovered that the Ada system had fewer comment lines than the FORTRAN system.

Table 3-5 presents productivity rates. Even with higher productivity, GRODY required more effort because of the larger size. This is seen by applying the rates of Table 3-5 to the raw sizes in Table 3-4. The Ada team would need between 32 and 42 percent more effort than would be needed to build a FORTRAN system with no reuse.

Q. **What was the change rate of software during the development (specifically, change and errors due to unfamiliarity with Ada)?**

A. Figures 3-2 and 3-3 present the changes made to each system, arranged by the type of change and the phase in which the need for the change was

Table 3-5. Productivity

Excluding Training		
	GRODY	GROSS
Dev LOC per hour	6.4	2.5
Dev statements per hour	2.0	1.5
Excluding Training and Acceptance Test		
	GRODY	GROSS
Dev LOC per hour	6.4	3.4
Dev statements per hour	2.0	1.9

6019

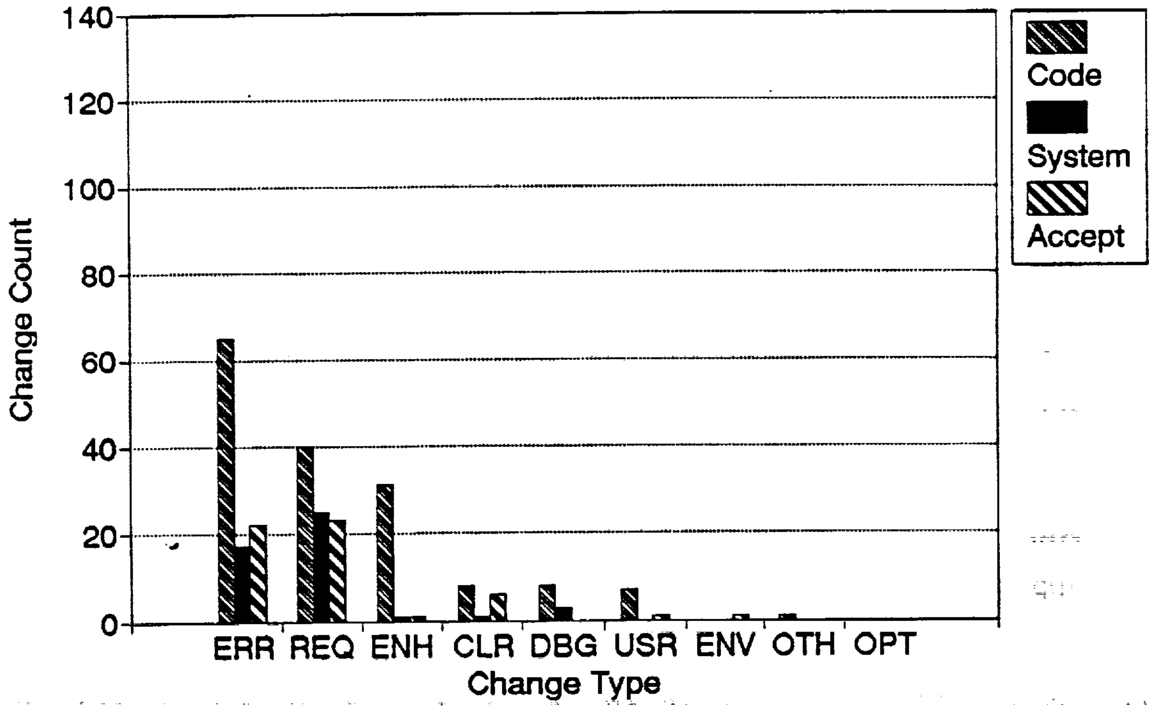

Figure 3-2. GROSS Changes by Type and Phase

6019

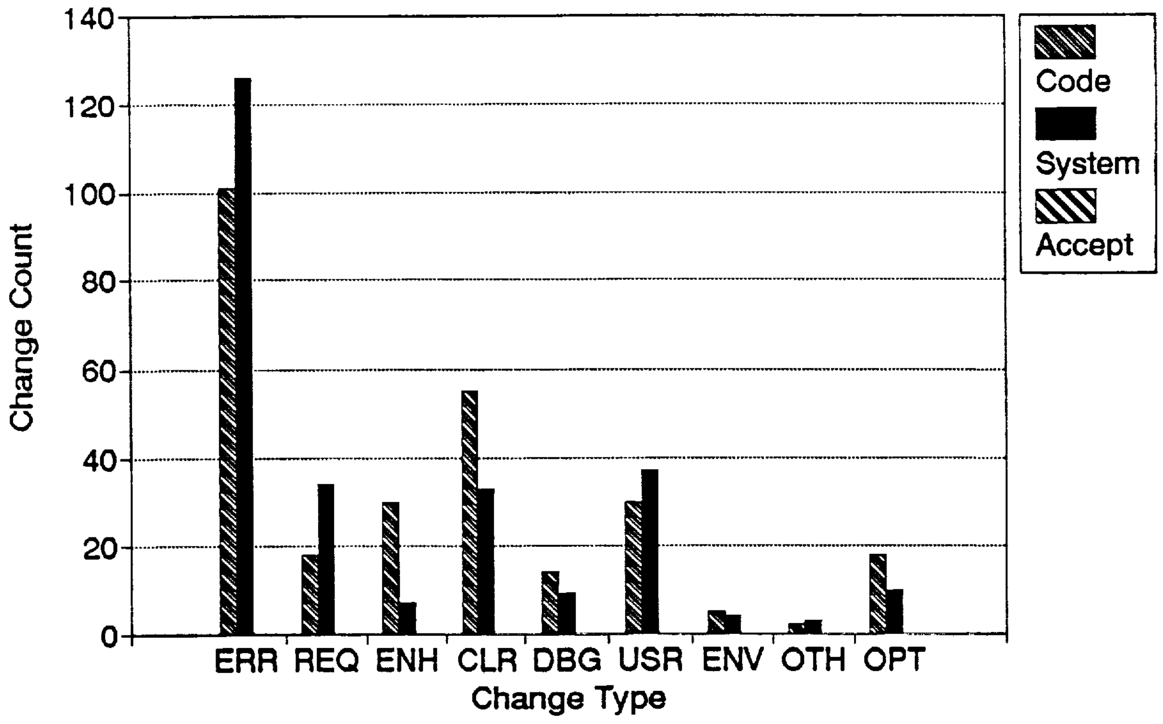

Figure 3-3. GRODY Changes by Type and Phase

determined. Two things are apparent: GRODY made more changes, and GRODY made changes for more diverse reasons. GRODY's top three types of change (error correction, enhancements for clarity and maintainability, and implementations of user services) all point to an exploratory project, especially when compared to GROSS. GROSS made nearly as many requirements changes as error corrections. GRODY recorded only a few changes due to Ada causes.

Q. **What was the relative effort required for making a change? Fix an error?**

A. Figures 3-4 and 3-5 present the effort required to make a change. Figures 3-6 and 3-7 present the effort required to fix an error. SEL records this data in two types of effort in four classes. The types are effort to isolate the change (error) and effort to complete the change (error). The four classes are

- Less than 1 hour
- More than 1 hour but less than 1 day
- Between 1 and 3 days
- More than 3 days

These figures show two items of note. First, GROSS tends to have a larger fraction of changes and errors that took 1 hour to isolate and 1 hour to

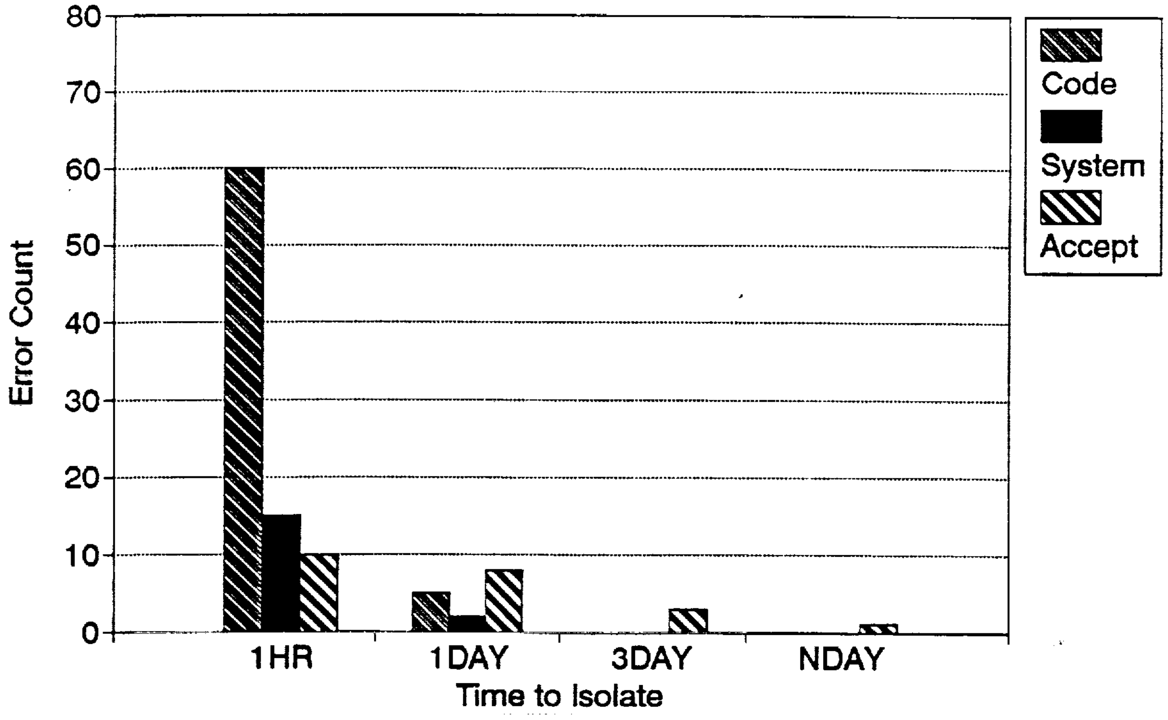

Figure 3-4. GROSS Effort To Isolate Errors

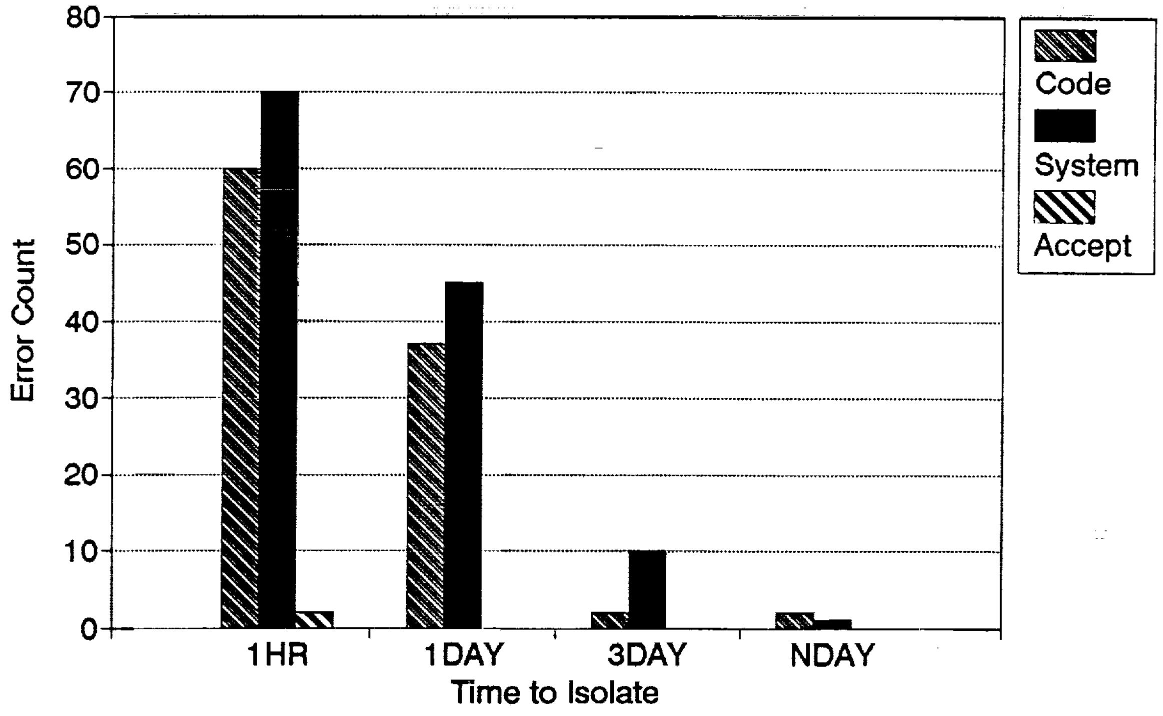

Figure 3-5. GRODY Effort To Isolate Errors

complete. Second, GROSS's severe changes and errors (the 3-day and N-day) tended to occur in acceptance testing. This is a likely effect of the protracted acceptance test schedule.

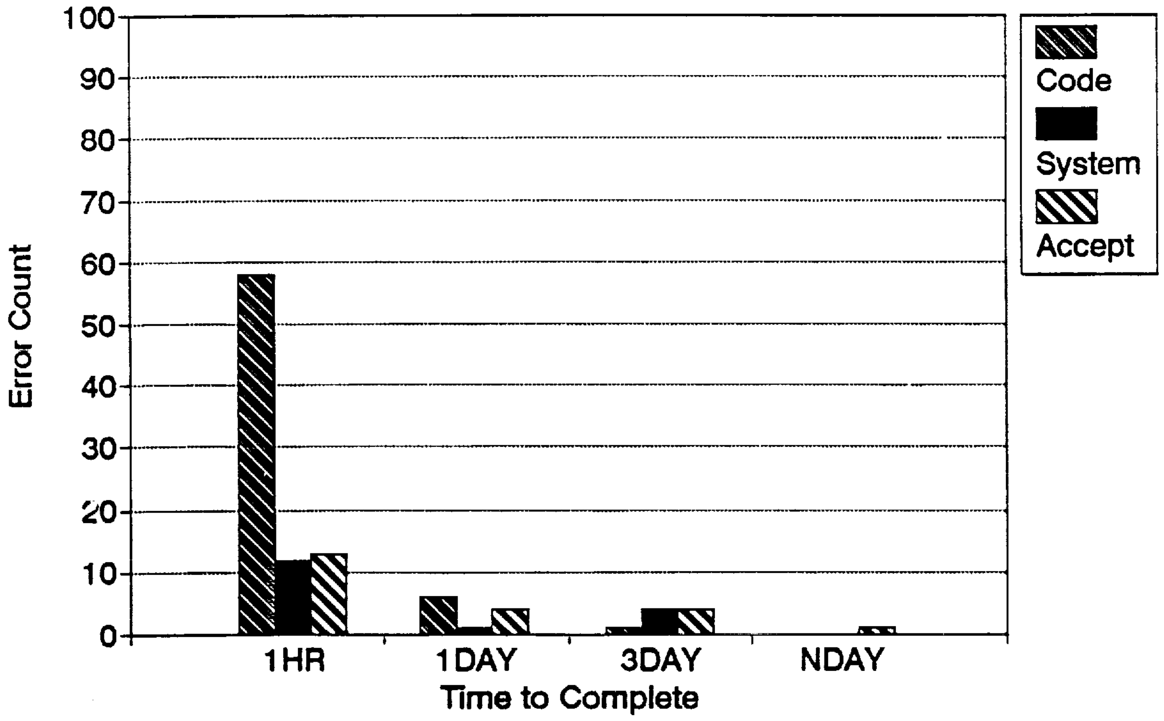

Figure 3-6. GROSS Effort To Repair Errors

Q. **How did the failures in the two developments differ in type and severity?**

A. The answer to the previous question addressed the severity of errors. Figures 3-8 and 3-9 illustrate the sources and classes of errors. Again, the pattern is seen of GROSS being well under control with the bulk of the errors coming from code, while GRODY is more exploratory. Note that GRODY had significant numbers of errors coming from design and previous changes. The classes of errors illustrate the GRODY team's lack of experience with both the application and Ada.

Q. **What is the relative effort required to implement a change in requirements?**

A. Figures 3-10 through 3-13 show that GROSS had more difficulty with requirements changes. However, two confounding factors exist. First, many of the changes occurred during acceptance testing, and it is not known how well the Ada team would have done with late requirements changes. Second, GRODY did not implement all the requirements changes that GROSS did. It is not known, on a change-by-change basis, the difficulty that each project had with each change. Consequently, firm conclusions can not be drawn about the effect that the implementation language had on implementing requirements

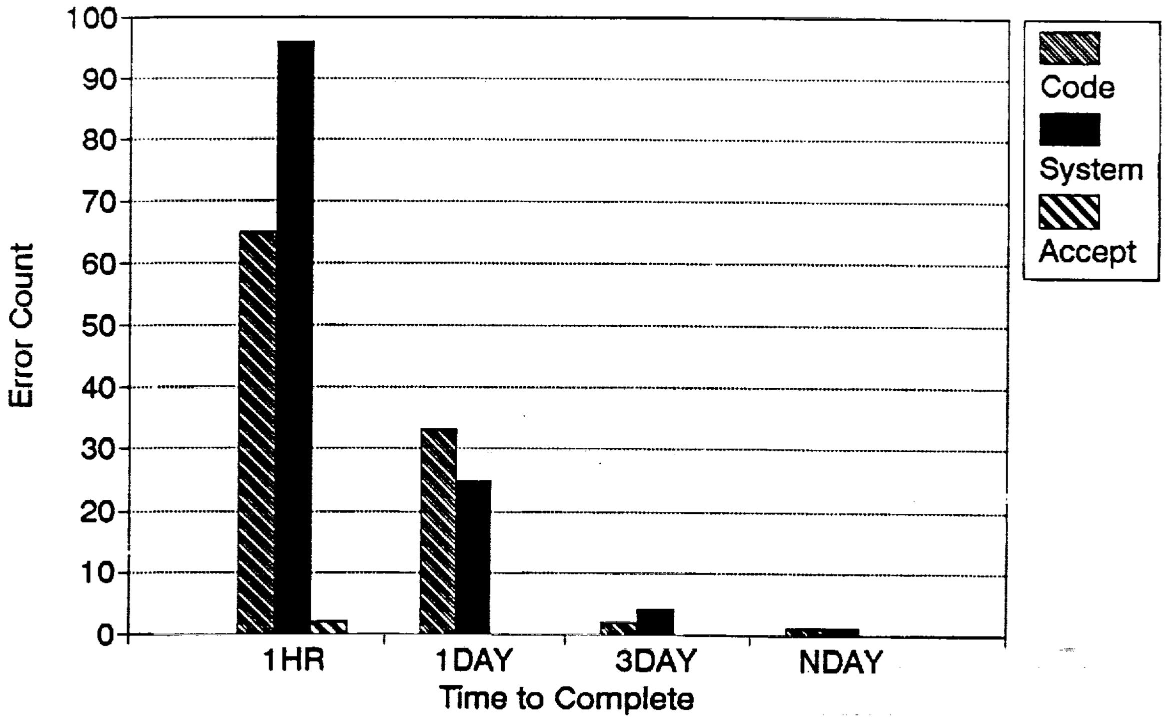

Figure 3-7. GRODY Effort To Repair Errors

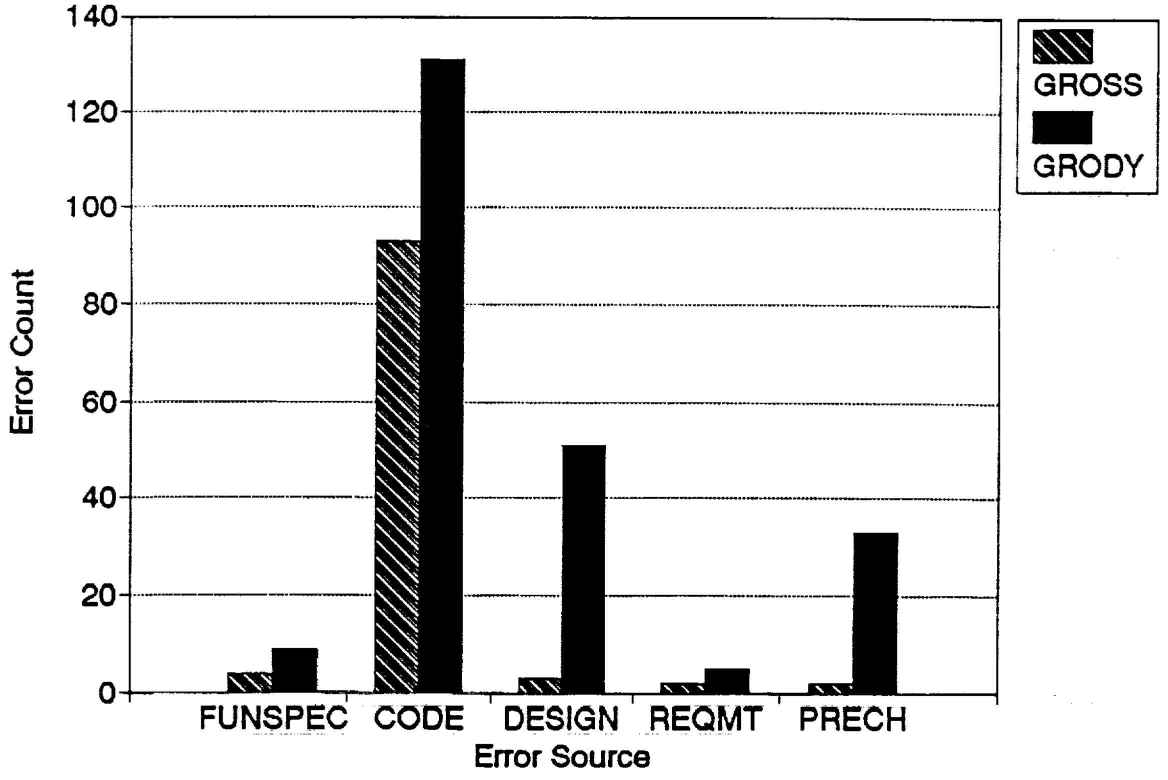

Figure 3-8. Sources of Errors

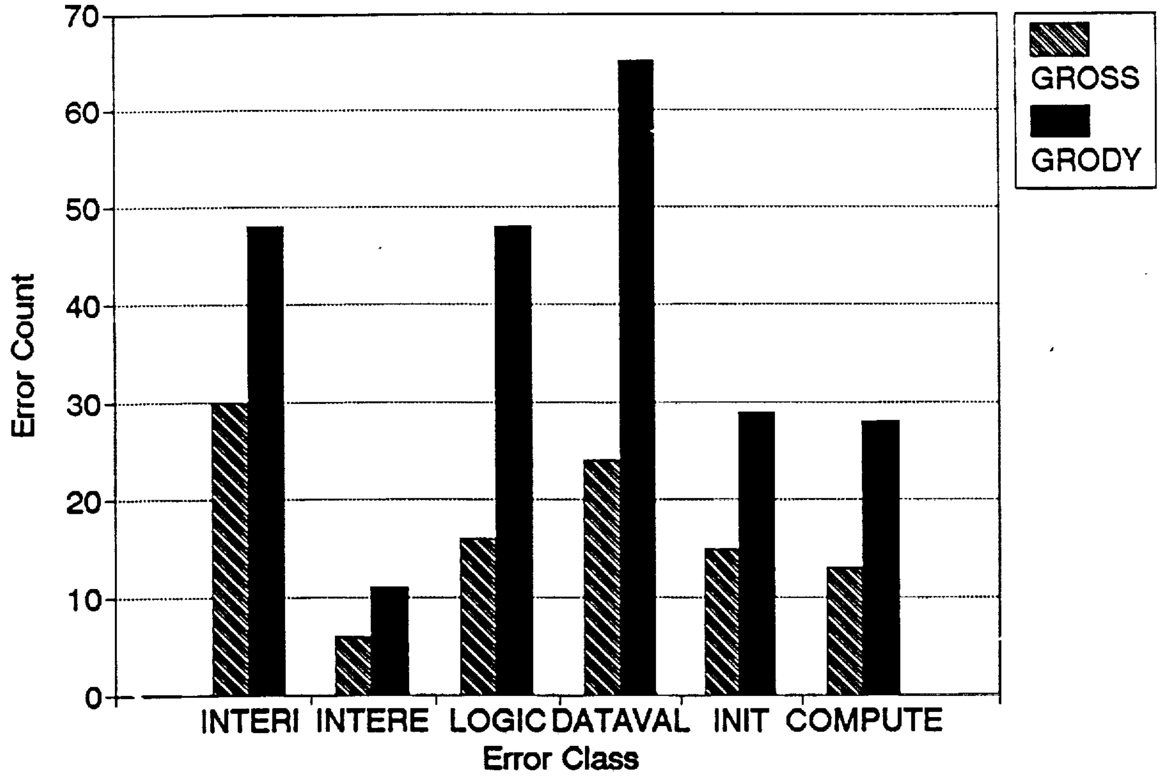

Figure 3-9. Classes of Errors

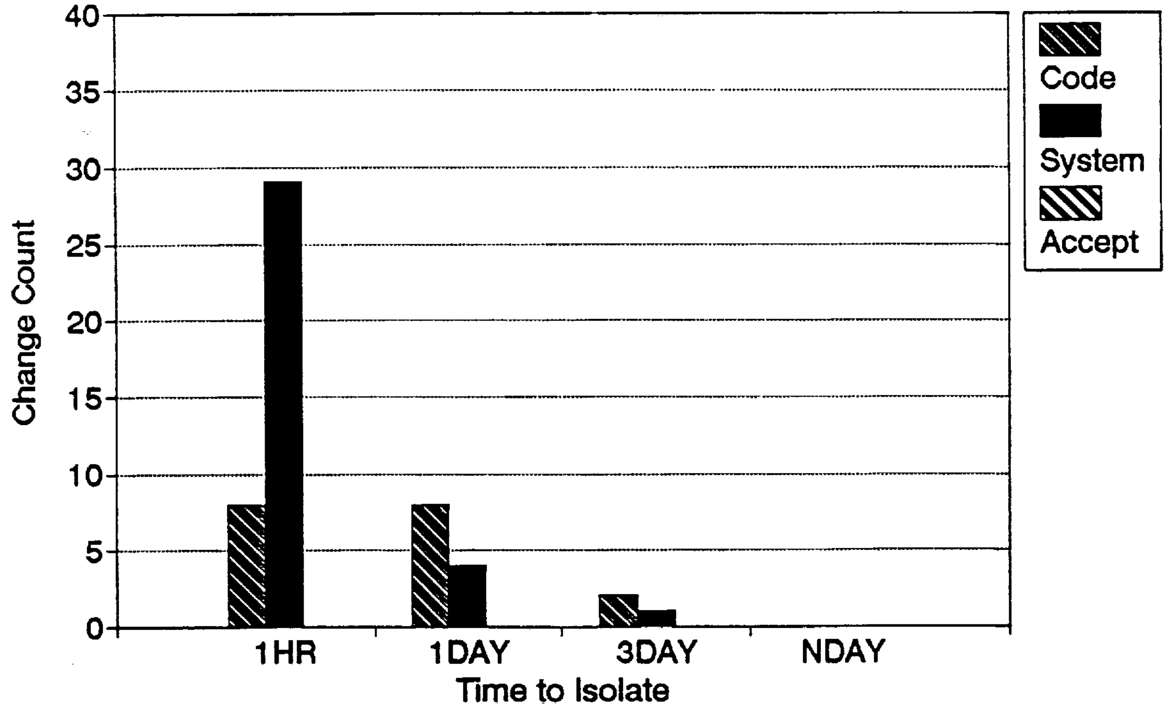

Figure 3-10. GRODY Effort To Isolate Requirements Change

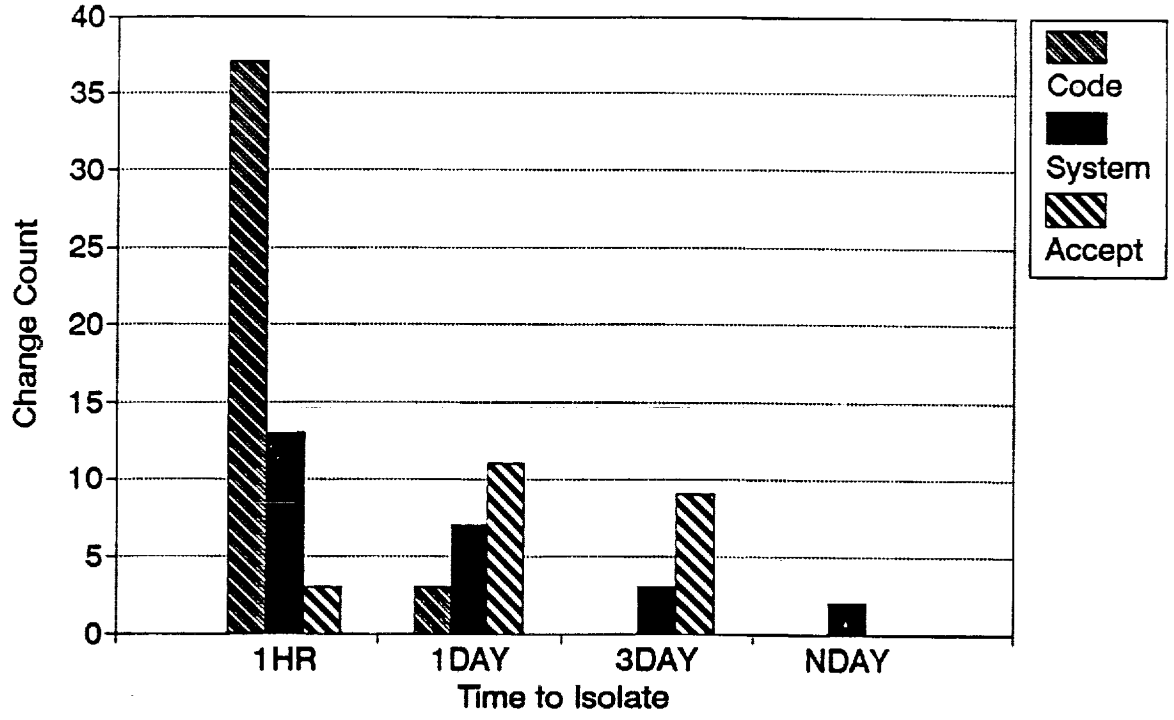

Figure 3-11. GROSS Effort To Isolate Requirements Change

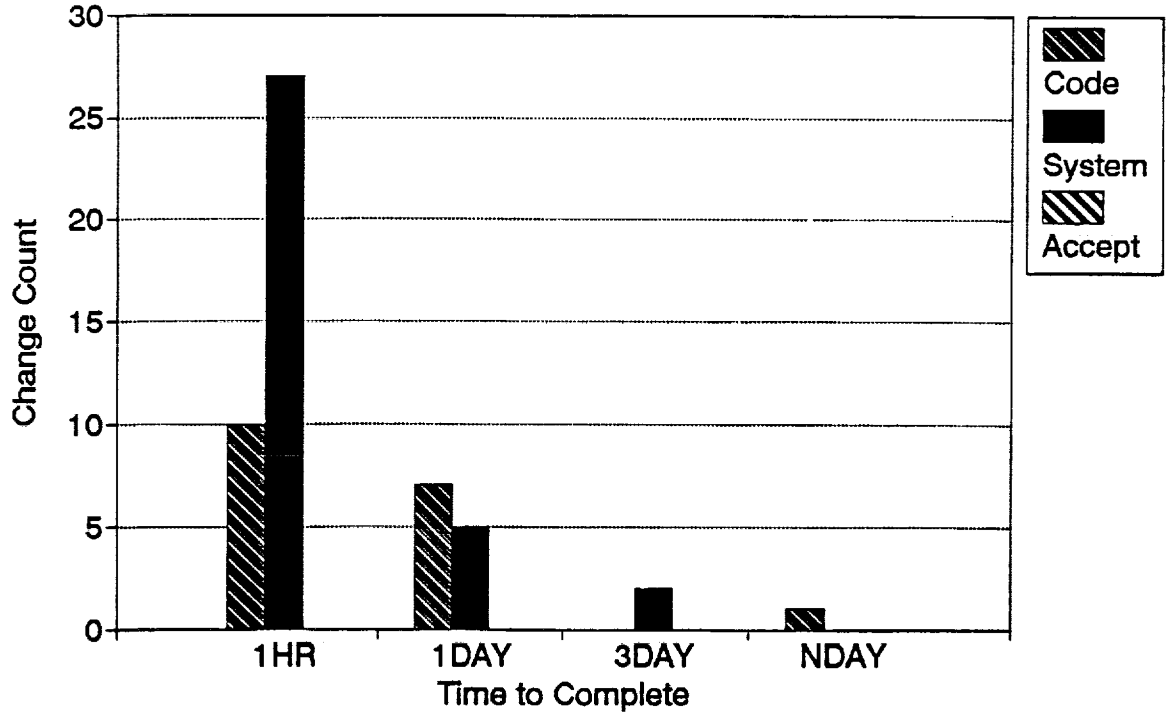

Figure 3-12. GRODY Effort To Implement Requirements Change

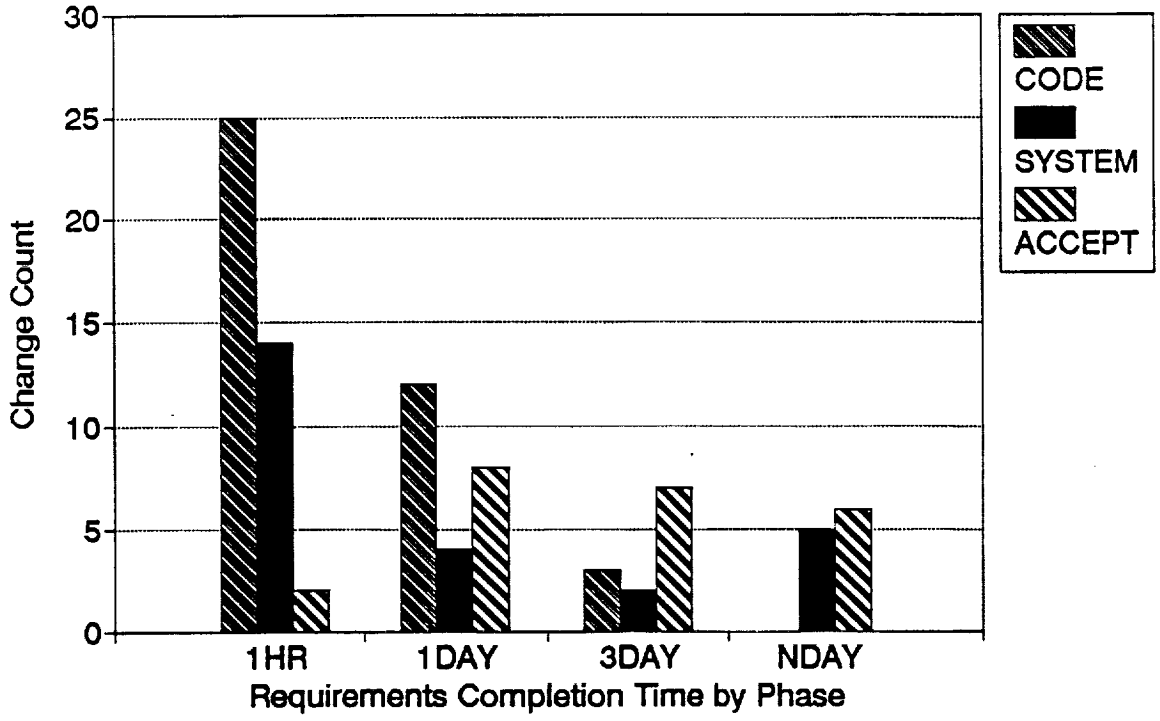

Figure 3-13. GROSS Effort To Implement Requirements Change

changes. It can be noted that these results do not reflect only the differences in the languages, but also large differences in the nature of the two projects.

APPENDIX A—AN EXPERIMENT WITH ADA—THE GRO DYNAMICS SIMULATOR PROJECT PLAN (APRIL 1985)

A.1 PROJECT DESCRIPTION

The overall goal of the Ada/FORTRAN software development project is to develop insight into the applicability of the Ada development methodology and language into the NASA software environment. To determine the potential application of Ada, a set of measures, objectives, questions, and detailed data must be defined.

A.1.1 Objectives

The following objectives have been developed as a mechanism toward attaining the overall goal stated above:

- Determine the cost-effectiveness of adapting Ada to specified environments (e.g., NASA ground systems)
- Develop a set of software measures useful in determining the advisability of adapting Ada to portions of the Space Station project
- Develop tradeoffs and characteristics of the potential impact of using Ada as development methodology and language (e.g., training, standards, etc.)
- Determine impact of Ada on the reuse of software (reusable software concepts)
- Develop and assess profiles (characteristics) of the development process using Ada as opposed to using classical high-order languages
- Determine effect of Ada (methodology and language) on
    1. Productivity
    2. Reliability
    3. Maintainability (effort to change and effort to repair)
    4. Reusability
    5. Portability

A.1.2 Questions To Be Addressed

To attain the objectives defined, a series of questions must be addressed. A list of the questions to be raised follows:

Cost Effectiveness

1. What is the relative manpower cost of designing, coding, testing, changing Ada software vs. FORTRAN software?

2. Excluding training, how does the staff effort for the Ada project differ from the FORTRAN project?

3. Does the machine expense differ?

4. Did the Ada project meet its deadlines? (FORTRAN project?)

5. How expensive was the Ada training? Could it be made more cost-effective?

6. How did the productivity on the two projects differ?

Measures for Space Station

1. Are the measures for determining quality and cost effectiveness of Ada identical to or similar to the high-order language measures?

2. Are there measures to determine the application of Ada to varying environments typical of NASA (size, machine type, application type, etc.)?

3. Can the Ada methodology be applied and assessed even if the language were not used for implementation?

4. How can the 'readiness' of Ada environment, including computers, tools, and methodologies, be determined?

5. How are training requirements determined in Ada? Vis-a-vis, those for other candidates (HAL/S, C, etc.)

6. Can the impact of 'deleting' software heritage in specific environments be assessed (reused design, code, etc.)?

7. Are there metrics that effectively distinguish between functional and object-oriented design and display the advantages of using either approach?

8. Are there metrics that can be applied to both Ada and FORTRAN projects so that they can be compared directly?

9. Which metrics best predict the effort that will be necessary to produce the Ada system?

Characteristics/Tradeoffs

1. Are people with certain types of academic backgrounds and professional experiences more readily adaptable to Ada? (What types?)

2. How difficult is the transition between Ada environments for the programmer?

3. Is the documentation for the Ada environments adequate enough that the users do not need to contact an Ada expert for help?
4. Would it be possible, and reasonable, to develop an Ada system without an Ada consultant onboard for help?
5. Was the training sufficient or was it supplemented as problems developed? From the experience, can a sufficient training program be developed?
6. How should the training be improved?
7. What features must be included in an Ada environment if it is to be used by NASA?
8. Which features in an Ada environment are least used at NASA?

Reuse of Software

1. How much of the design and code could be used from GRO to develop a simulator for ERBS (or any new simulator)?
2. Were the packages designed to be reused?
3. Were the specifications defined in sufficient generality?
4. Were generics used appropriately?
5. Were implementation details hidden?
6. In retrospect, would the designers create some parts to be more readily reused?
7. If they were developing a similar system in Ada, how much of the system would the designers reuse?
8. Is the completed Ada code more readable and easier to comprehend than other languages?

Profiles of Development

1. What is the relative effort expended in training, design, code, testing, etc.?
2. What computer resources were used compared to the FORTRAN project?
3. What was the change rate of software during the development (specifically, change and errors due to unfamiliarity with Ada)?
4. What was the error rate for each system?

5. What was the relative effort required for making a change? Fix an error?

6. How did types of faults differ in the systems?

Effect on Productivity

1. What was the relative cost of each phase?

2. In the delivered product, were there more or fewer failures using the Ada system?

3. How did the failures in the two developments differ in type and severity?

4. How much effort would be needed to transport the system to another machine with the same Ada environment? (Different environment)

5. What is the relative effort required to implement a change in requirements?

A.1.3 Data To Be Collected

To respond to the questions raised in Section A.1.2, data must be collected from both development teams as well as from other sources (source code, design, managers, etc.). The data to be collected are defined in Section A.4.2.

A.2 PROBLEM DESCRIPTION

A.2.1 Description

The overall function of the simulator is to be able to receive ground commands and environmental information and, in turn, to model the performance of the onboard logic that is attempting to maintain some specified spacecraft attitude by generating commands for actuators and thrusters.

The dynamics simulator required to support the GRO Project will satisfy several major functions. First, the simulator is needed to validate software that is to fly onboard GRO. Second, it is needed to study the performance of the attitude control system onboard and to train operational control center personnel by simulating responses to particular commands and by simulating responses to various spacecraft phenomena, such as a failed sensor or actuator.

The simulator is given all the physical characteristics of GRO and then is required to model the dynamics of the satellite, given a varying set of conditions. Typical dynamic simulators have five major components:

1. Model of the onboard computer (OBC) - Attitude Control System

2. Sensor and spacecraft model (truth models)

3. Environment model (torques and ephemerides)

4. Input/Output Control System

5. Postprocessor

This particular project is similar in concept to the one developed to support the ERBS mission (launched in 10/84), except the ERBS did not have an OBC (it had an analog control system) while the GRO mission is flying the NASA Standard Spacecraft Computer (NSSC-1). Both simulators are required to perform functions described by the listed five major components.

A.2.1.1 OBC MODEL

This portion of the system simulates the functions that are performed by the flight computer (or control system), the NSSC-1. It must produce actuator commands and telemetry information, given sensor readings and ground commands.

A.2.1.2 SENSOR AND SPACECRAFT MODEL - (TRUTH MODEL)

This component is required to model the response of the spacecraft hardware (e.g., sensors) based on ephemeris data, environmental data (e.g., magnetic field readings), and current state and commands for the OBC.

A.2.1.3 ENVIRONMENTAL MODELING (PROFILE)

The major function of this portion is to determine the torques acting on the spacecraft based on normal attitude and orbit information as well as spacecraft characteristics. This portion also determines the ephemeris and magnetic field information as well as star positions.

A.2.1.4 INPUT/OUTPUT CONTROL SYSTEM (SCIO)

This set of requirements defines the controlling function for both the truth model and the OBC. This system is also the interface and control with the user.

A.2.1.5 POSTPROCESSOR

It is a requirement to produce various CRT displays as well as plots and some statistical information from a simulator run. In past systems, this was handled as a separate function in the simulator.

A.2.2 Size Characteristic

Based on previous projects, an estimate of the full system, when developed in FORTRAN, follows:

	(Lines of FORTRAN Code with Comments)
Modeling of onboard processing	10,000
Truth model	12,000
Profile program	6,000
Input/output control	10,000
Postprocessor	2,000

A.2.3 Requirements Development

The functional requirements for the GRO Dynamic Simulator are developed by the Flight Dynamics Analysis Branch (Code 554) and are delivered to the Systems Development Branch (Code 552) for implementation. After delivery, the requirements are considered to be under configuration control, and any modifications must go through a formal change process followed by both branches.

A.2.4 Responsibility of Development Team

The implementation team is given the functional requirements and then is responsible for carrying out the full development of the software. The team is responsible for design, code, test certification, and documentation. The final product is typically delivered to the Flight Dynamics Analysis Branch; but for this project, an internal acceptance test and delivery will be made.

A.3 PROJECT MANAGEMENT STAFFING

Personnel from three distinct installations–NASA/GSFC, CSC, and the University of Maryland will support this ADA experiment. Three functional teams or groups also will be part of this project: the Ada development team, the FORTRAN development team, and the study group directing the experiments.

A.3.1 Team Structure

A.3.1.1 ADA DEVELOPMENT TEAM

The team responsible for the design, implementation, and test of the dynamics simulator using the Ada development methodology and language will consist of approximately seven software developers. Each developer will be devoting a minimum of 1/3 time, with an average of 1/2 time allocated by each person. These resources are expected to be provided from the following organizations. (It is estimated that the project will continue at least through July 1986.)

- Code 520—Two developers, each allocating between 1/2 to 3/4 time. These people will charge to a Code 520 charge account.
- Code 550—Two developers, each allocating between 1/2 to 3/4 time. These people will charge to the Code 550 Code T RTOP.
- CSC (via Task 455)—Three developers, each allocating 1/2 to 3/4 time. These charges will be covered by Task 455, which is funded and managed by Code 552. CSC will also participate in funding for half this effort.

The *management* of the Ada implementation will be as follows:

- Overall project management—McGarry (550)/Nelson (520)

Responsibilities include the allocation of resources (including manpower, funding, computer resources, etc.), setting of schedule/milestones, coordination of requirements availability, and general interface to all other organizations and teams. Also responsible for coordinating the measures and general experimental information required.

- Technical Direction—Agresti (CSC)

Responsible for specific implementation of the problem using the Ada approach. Makes specific technical assignments to team members, reviews specific work (code, design, testing, etc.), takes an active part in the design, code and general development and, in general, acts as Chief Programmer of the team.

- Training and Methodology Development—Basili (University of Maryland)

Responsible for developing all required training and for directing the overall methodology to be followed by this team.

Additional *consultation* support will also be available. Through a support contract, Code 520 will provide additional Ada expertise (Ada Soft Inc.) for use by the development team. This consultation will be available to review designs, development approaches, and general problems with Ada. The extent of their availability will be determined by Code 520.

A.3.1.2 FORTRAN DEVELOPMENT TEAM

The team responsible for the implementation of the FORTRAN version of the dynamics simulator will be comprised of personnel from CSC and from Code 550. The project will be handled exactly as any other development effort under the PC&A contract. The implementation team will be comprised of the following:

Code 550—3 developers, each allocating from 1/2 to full time on the project.

CSC—3 to 4 developers, each allocating from 1/2 time to full time.

The *management* of the FORTRAN implementation:

- Overall Project Management—Behuncik/Garrick (550)
- Technical Direction—Garrick/550 and Task Leader at CSC

It is not anticipated that any special interface or contact will be required because of the nature of the experiment on the FORTRAN team. There need to be no interface with either the University of Maryland or with Code 520. It is expected that SEL data will be provided, as usual, by the FORTRAN team.

This effort will be completely funded and controlled by Code 550.

A.3.1.3 ADA EXPERIMENT STUDY GROUP

The team responsible for defining, directing, and analyzing the overall Ada project experiment will consist of staffing from GSFC, the University of Maryland, and CSC. This team will be responsible for defining the overall experiment (including measures, goals, training required, data to be collected) as well as to analyze results. Specifically, the team will

- Develop plans, goals, objectives, and data to be allocated
- Define the procedures for carrying out the project
- Develop and write the overall project plan
- Develop and provide (or make available) all required training and project preparations
- Monitor project development and recommend any required adjustments (such as staffing, resources, training)
- Complete analysis of the project and develop report(s) describing these results

This team will consist of the following staff:

GSFC—Code 550	F. McGarry (1/10 time)
GSFC—Code 520	R. Nelson
Univ. of MD	V. Basili (1/10 time) E. Katz (1/2 time)
CSC	W. Agresti (1/10 time)

A.3.2 Resources

Sources of funding to cover required staffing, training, computer support, etc., will include GSFC (Codes 520 and 550) and CSC.

A.3.2.1 STAFFING

A.3.2.1.1 Ada Development Team

- Code 520 Support—All 520 costs (technical developers and managers) will be charged to a Code 520 job order number.
- Code 550 Support—All 550 in-house costs will be charged to the Code T RTOP, Software Engineering (310-10-23)
- CSC Support—Task charges will be covered by the Code 550 task, which is paid by the Code T RTOP (310-10-23). CSC will also pay for half the manpower required by the Ada development team.

A.3.2.1.2 FORTRAN Development Team

The entire FORTRAN staffing effort will be the responsibility of Code 550. Internal costs will be charged to a Code 550 JON, and all CSC costs will be charged to the Code 550 task.

A.3.2.1.3 Ada Experiment Study Group

- Code 550 Support—Support will be charged to the Code T RTOP 310-10-23.
- Code 520 Support—Support will be charged to a Code 520 JON.
- CSC Support—All CSC charges will be covered by the Code 550 task for the project.
- University of MD—All charges will be covered by the Code 550 grant with the University of MD (NSG 5123).

A.3.2.2 COMPUTER SUPPORT

- To support the development efforts of the Ada project, computer support will be supplied in the following manner:
- VAX 11/780 Development Machine—Time, terminals, and necessary access will be supplied by the Code 520 VAX and by the Code 550 VAX. All team members are to have access to either machine on a continuing basis. Accounts are to be provided to all team members and managers on both computers. For the processors not normally scheduled to be in operation during off-hours, special requests may be made to access the processor at nights and/or weekends. Computer time expense will be completely covered by the respective divisions.
- Compilers/tools—Both VAX machines will house the same Ada compiler and support environment.

The DEC compiler will be purchased by Code 520 for its VAX and by Code 550 for its VAX.

A.3.2.3 TRAINING

Expense of training team members will be divided as follows:

- Ichbiah Tapes—To be paid for by and then provided by Code 520.
- Computer Thought (Hypergraphics) PC disks—To be made available to entire team and to be paid through NASA centerwide funding.
- Informal/Interactive Training—Will be supplied by University of Maryland personnel and will be covered through the Software Engineering grant #NSG5-123.

A.3.3 Team Interaction

There will be a close working relationship between the Ada development team and the Ada experimenter team. This will include periodic meetings to review status, difficulties, etc., in an attempt to identify additional training needed or to identify and help in certain problems with the Ada language or methodology.

Although there will be no attempt to completely isolate the FORTRAN team from the Ada development team, there will be no attempt to encourage interaction. Formal procedures will be defined whereby requirements change, errors, and general modifications will be made known to both teams in the same timeframe and format. No attempt will be made to provide questions/answers generated by one team to the other team. The designs, development methodologies, and general procedures of one team will not be reviewed by the other team.

The only interaction between the FORTRAN team and the Ada experimenters will be for the experimenters to periodically review the data and information being recorded by the FORTRAN team

A.4. DEVELOPMENT PLAN

To be able to respond to goals and questions addressed in Section A.1, the process by which the development team will be prepared and the process that will be in effect during the development period must be defined.

The specific plan of carrying out the development activities will address training, data collection, development standards/approach, and configuration control.

A.4.1 Training

As pointed out in reports describing other Ada experimental efforts, training of the development team is both very critical and very difficult. Responding to experiences of other projects attempting to use Ada, an extensive effort is to be put forth to train the Ada team. The training activities will be broken into four phases:

- Phase 1 (Initial Training)—Approximately a 2-week period will be allotted for the first phase of training. All team members will be provided a comprehensive text on the Ada methodology ('Software Engineering with Ada' by G. Booch) and will be assigned some preparatory reading assignments before class meetings and initiated.

After a basic introduction, a 1-week class will be held (full days) where video tapes (Ichbiah) will be used as instruction material. The 23 tapes are to be reviewed and discussed. The class leader also directs discussions and assigns sample problems for reinforcement.

- Phase 2—Approximately 3 weeks into the project, all class members are to attend a formal 3-day course where the overall methodology of Ada is taught. The course is taught by G. Cherry.

- Phase 3—After the 3-day methodology course, the development team will work with at least one practice problem where they will be required to design, implement, and test the problem. This will be done under the supervision of Ada instructor from the University of Maryland (Beth Katz, Vic Basili). Because the development team will need to begin requirements analysis as well as gain access to a changing Ada environment, it is anticipated that this further training will last appropriately 2 to 3 months.

- Phase 4—After the design and implementation of the simulator is started, a continued reinforcement of those points covered in the training classes will take place. It is anticipated that on an as-needed basis, the team will meet for additional training under the direction of the University of Maryland. This training will last for the duration of the project.

A.4.2 Data Collection

Detailed information will be collected in several forms so that the overall experiment may be accurately studied. The following is a description of

- Data to be collected
- Data QA and processing procedures

A.4.2.1 DATA COLLECTION FORMS

- Programmer/Analyst Survey—All programmers, managers, and librarians will complete this form as soon as they begin work on the project. This includes the Ada team and the FORTRAN team.

- Resource Summary Form—This form will be completed on a weekly basis, beginning on January 1, 1985, by all team members and managers. Total weekly hours expended on the Ada project are recorded. Computer resources will be provided by the librarian through the VAX accounting.
- Component Status Report—This form will be completed on a weekly basis beginning January 1, 1985, by all team members. Personnel devoting 100 percent of their project time to 'management' will not complete this form.
- Component Origination Form—This form will be completed whenever a particular 'component' of the project is identified (design). An Ada component is defined as a task, package, or subprogram. The person first identifying the component is responsible for completing the form.
- Change Report Form (1 and 2)—Both the standard SEL change report form, as well as the added page of questions will be completed by the Ada team. The forms are to be completed for any change/error defined after a programmer has completed the component origination form. The form will be completed by the individual making the change.
- Project Estimation Form—This form will be completed by the project technical manager every 6 weeks beginning in early January. The form may be completed in consultation with other team members.
- Project Header File Information Form—This form contains all actual sizes and characteristics of the completed project. It will be completed once at project completion by the technical manager.
- Additional Data—In addition to the forms for collecting data, two additional sets of information will be saved.
- Change/Growth History—Each week, the support librarian will record the number of lines of code, number of components, and total number of changes that have been made to the source library.
- Subjective File Data—At the completion of the project, a set of parameters that characterizes the development process and product will be determined by the managers of the development effort.

A.4.2.2 DATA QA AND PROCESSING

Forms are to be completed and turned into a designated individual within each organization. These individuals are responsible for both reminding individuals to provide the data and for doing the first-level QA.

Designated individuals are

CSC	-	Bill Agresti
GSFC (520 and 550)	-	Betsy Edwards
University of MD	-	Beth Katz

All forms are then to be forwarded to the GSFC coordinator (Betsy) for the next level of QA. Finally, forms are turned over to the librarian support for entry and final QA (Barbara and Keith).

The hardcopy is also stored in the SEL library.

A.4.2.3 DEVELOPMENT STANDARDS/APPROACH

The general procedures and standards to be followed by the development team are called out in the "Recommended Approach to Software Development." This document defines such practices as walkthroughs, use of librarian, PDL, and inspections. Modifications to these guidelines will be defined by the project management team so that they are not in conflict with the Ada development methodology.

A.4.2.4 CONFIGURATION CONTROL

Both online tool support as well as offline support will be provided for the Ada development effort. If the DEC-supported Component Management System/Module Management System (CMS/MMS) provide support to the Ada library, these tools or their equivalent will be adapted for use.

In support of the development project, the configuration management process will be developed and defined by the technical team. They will define procedures for generating, maintaining, changing, and moving libraries associated with the project. It will be the responsibility of the technical leader to see that defined procedures are carried out.

A.5 SCHEDULES AND PRODUCTS

Although the project is tasked with developing a working software product as a major end-item, the most critical end-item to be developed by the experiment is the record of experiences and the assessment of Ada as a potential development methodology and development language.

A.5.1 Products To Be Completed

The overall products to be completed and delivered for this project may be considered to be in three categories:

1. Software products
2. Forms and data
3. Reports and analysis

A.5.1.1 SOFTWARE PRODUCTS

Although the Ada team possibly may be unable to successfully complete all software to the point where a completely integrated, verified simulator exists, the target still remains to have both teams complete the GRO Dynamics Simulator. Each team will produce the following software products:

- Software development plan
- Requirements analysis report
- Design report
- Completed source code
- Software test plan
- Software development history report

In addition, the *FORTRAN Team** will produce:

- Simulator user's guide
- Simulator system description

Formats, timeline, and contents of these reports and documents are defined in the SEL document "Managers Handbook for Software Development."

A.5.1.2 FORMS AND DATA

Both teams are responsible for generating detailed histories of the development process by completing the Software Engineering Forms. The description of this data is found in Section A.4.2.1.

All of the collected derived data will be processed by SEL personnel and then stored on the SEL data base.

*It will be decided at a later date (July 1986) whether or not the Ada team will produce these two documents.

A.5.1.3 REPORTS AND ANALYSIS

Because of the magnitude of this experiment, it is anticipated that an extensive set of reports and papers will be generated by and for the participants in the experiment. It is the responsibility of the experimenter team to assure that key reports supporting the analysis of the experiment are completed either by development team members (Ada and/or FORTRAN) or by the experimenter team.

Although, at this time, the total list of reports and papers that may be generated by this experiment cannot be developed, the following list of reports is considered a minimum set:

- "An Experiment with Ada in the SEL" (full SEL document)
- "An Evaluation of Training Approaches for Ada"

- "An Assessment/Experience with Object-Oriented Design and Process Abstraction for Software Design"
- "Implications of Ada Methodologies on the Reuse of Software"
- "An Assessment of Ada as Applied to NASA Ground System's" (possibly part of Bob Murphy's PIP)
- "Comparing Characteristics of the Development Life Cycle for an Ada and a FORTRAN Project"
- "Measuring the Potential of Ada for the Space Station Project" (probably Bob Nelson's responsibility)

Additional reports will be defined during and after the overall experiment is completed. It is anticipated that all members of the Ada team as well as all members of the experimenter team will be responsible for participating in writing portions of the above reports.

All reports will either become full SEL documents or will be incorporated into the series of SEL reports, "Collected Papers...."

One of the major portions of the study will be incorporated into the Ph.D. dissertations of Beth Katz. This may later become a portion of the full SEL report describing the overall experiment and results.

A.5.2 Schedules and Milestones

Key dates for the experiment are presented as follows.

A.5.2.1 ADA TEAM

• Project start	1/1/85
• Training complete	4/1/85
• Software development plan	5/1/85
• Requirements analysis	5/1/85
• Complete design	11/1/85
PDR	7/1/85
CDR	11/1/85
• Complete code/unit test	7/1/86
• Complete integration test	10/1/86

• Software test plan	5/1/86
• Software development history	12/1/86

A.5.2.2 FORTRAN TEAM

• Project start	2/1/85
• Software development plan	1/15/85
• Requirements analysis	3/15/85
• Complete designs	
PDR	4/11/85
CDR	6/1/85
• Complete code/unit test	1/1/86
• Complete integration test	5/1/86
• Software test plan	12/1/85
• Complete acceptance test	10/1/86
• User's guide	1/1/86
• System description	1/1/86
• Software development history	12/1/86

A.5.2.2 GENERAL SCHEDULES

• DEC compiler available on 550 VAX	5/85
• DEC compiler available on 520 VAX	5/85
• Reports and documents	
1. "Assessment of Ada Training Approaches"	7/1/86
2. "An Experiment with Ada in the SEL"	12/85
3. Other reports - TBD	

APPENDIX B—GRODY PUBLICATIONS

Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team, SEL-85-002, R. Murphy and M. Stark, October 1985

"Designing With Ada for Satellite Simulation: A Case Study," W. W. Agresti, V. E. Church, D. N. Card, and P. L. Lo, *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

"Towards a General Object-Oriented Software Development Methodology," E. Seidewitz and M. Stark, *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

General Object-Oriented Software Development, SEL-86-002, E. Seidewitz and M. Stark, August 1986

"Towards a General Object-Oriented Ada Life-cycle," M. Stark and E. Seidewitz, *Proceedings of the Joint Ada Conference*, March 1987

"Lessons Learned in Use of Ada™-Oriented Design Methods," C. E. Brophy, W. W. Agresti and V. R. Basili, *Proceedings of the Joint Ada Conference*, March 1987

Ada® Style Guide (Version 1.1), SEL-87-002, E. Seidewitz, May 1987

Assessing the Ada® Design Process and Its Implications: A Case Study, SEL-87-004, C. Brophy and S. Godfrey, July 1987

"Measuring Ada for Software Development in the Software Engineering Laboratory (SEL)," F. E. McGarry and W. W. Agresti, *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

"General Object-Oriented Software Development: Background and Experience," E. Seidewitz, *Proceedings of the 21st Hawaii International Conference on System Sciences*, January 1988

"Lessons Learned in the Implementation Phase of a Large Ada Project," C. E. Brophy, S. Godfrey, W. W. Agresti, and V. R. Basili, *Proceedings of the Washington Ada Technical Conference*, March 1988

"General Object-Oriented Software Development With Ada: A Life Cycle Approach," E. Seidewitz, *Proceedings of the Case Technology Conference*, April 1988

"Experiences in the Implementation of a Large Ada Project," S. Godfrey and C. Brophy, *Proceedings of the 1988 Washington Ada Symposium*, June 1988

System Testing of a Production Ada Project: The GRODY Study, SEL-88-001, J. Seigle and Y. Shi, November 1988

Implementation of a Production Ada Project: The GRODY Study, SEL-89-002, S. Godfrey and C. Brophy, May 1989

Lessons Learned in the Transition to Ada From FORTRAN at NASA/Goddard, SEL-89-005, C. Brophy, November 1989

GLOSSARY

CDR	critical design review
CRT	cathode ray tube
CSC	Computer Sciences Corporation
CSM	Composite Specification Model
ERBS	Earth Radiation Budget Satellite
FDD	Flight Dynamics Division
GOOD	General Object-Oriented Development
GRO	Gamma Ray Observatory
GRODY	GRO dynamics simulator in Ada
GROSS	GRO dynamics simulator in FORTRAN
GSFC	Goddard Space Flight Center
I/O	input/output
NASA	National Aeronautics and Space Administration
NSSC	NASA Standard Spacecraft Computer
OBC	onboard computer
PAMELA	Process Abstraction Method for Embedded Large Applications
PDL	program design language
PDR	preliminary design review
SEL	Software Engineering Laboratory
SLOC	source lines of code

STANDARD BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-ORIGINATED DOCUMENTS

SEL-76-001, *Proceedings From the First Summer Software Engineering Workshop*, August 1976

SEL-77-002, *Proceedings From the Second Summer Software Engineering Workshop*, September 1977

SEL-77-004, *A Demonstration of AXES for NAVPAK*, M. Hamilton and S. Zeldin, September 1977

SEL-77-005, *GSFC NAVPAK Design Specifications Languages Study*, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-005, *Proceedings From the Third Summer Software Engineering Workshop*, September 1978

SEL-78-006, *GSFC Software Engineering Research Requirements Analysis Study*, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, *Applicability of the Rayleigh Curve to the SEL Environment*, T. E. Mapp, December 1978

SEL-78-302, *FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 3)*, W. J. Decker and W. A. Taylor, July 1986

SEL-79-002, *The Software Engineering Laboratory: Relationship Equations*, K. Freburger and V. R. Basili, May 1979

SEL-79-003, *Common Software Module Repository (CSMR) System Description and User's Guide*, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, *Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment*, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, *Proceedings From the Fourth Summer Software Engineering Workshop*, November 1979

SEL-80-002, *Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation*, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-003, *Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study*, T. Welden, M. McClellan, and P. Liebertz, May 1980

SEL-80-005, *A Study of the Musa Reliability Model*, A. M. Miller, November 1980

SEL-80-006, *Proceedings From the Fifth Annual Software Engineering Workshop*, November 1980

SEL-80-007, *An Appraisal of Selected Cost/Resource Estimation Models for Software Systems*, J. F. Cook and F. E. McGarry, December 1980

SEL-81-008, *Cost and Reliability Estimation Models (CAREM) User's Guide*, J. F. Cook and E. Edwards, February 1981

SEL-81-009, *Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation*, W. J. Decker and F. E. McGarry, March 1981

SEL-81-011, *Evaluating Software Development by Analysis of Change Data*, D. M. Weiss, November 1981

SEL-81-012, *The Rayleigh Curve as a Model for Effort Distribution Over the Life of Medium Scale Software Systems*, G. O. Picasso, December 1981

SEL-81-013, *Proceedings From the Sixth Annual Software Engineering Workshop*, December 1981

SEL-81-014, *Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL)*, A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-81-101, *Guide to Data Collection*, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

SEL-81-104, *The Software Engineering Laboratory*, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

SEL-81-107, *Software Engineering Laboratory (SEL) Compendium of Tools*, W. J. Decker, W. A. Taylor, and E. J. Smith, February 1982

SEL-81-110, *Evaluation of an Independent Verification and Validation (IV&V) Methodology for Flight Dynamics*, G. Page, F. E. McGarry, and D. N. Card, June 1985

SEL-81-205, *Recommended Approach to Software Development*, F. E. McGarry, G. Page, S. Eslinger, et al., April 1983

SEL-82-001, *Evaluation of Management Measures of Software Development*, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-004, *Collected Software Engineering Papers: Volume 1*, July 1982

SEL-82-007, *Proceedings From the Seventh Annual Software Engineering Workshop*, December 1982

SEL-82-008, *Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory*, V. R. Basili and D. M. Weiss, December 1982

SEL-82-102, *FORTRAN Static Source Code Analyzer Program (SAP) System Description (Revision 1)*, W. A. Taylor and W. J. Decker, April 1985

SEL-82-105, *Glossary of Software Engineering Laboratory Terms*, T. A. Babst, F. E. McGarry, and M. G. Rohleder, October 1983

SEL-82-806, *Annotated Bibliography of Software Engineering Laboratory Literature*, M. Buhler and J. Valett, November 1989

SEL-83-001, *An Approach to Software Cost Estimation*, F. E. McGarry, G. Page, D. N. Card, et al., February 1984

SEL-83-002, *Measures and Metrics for Software Development*, D. N. Card, F. E. McGarry, G. Page, et al., March 1984

SEL-83-003, *Collected Software Engineering Papers: Volume II*, November 1983

SEL-83-006, *Monitoring Software Development Through Dynamic Variables*, C. W. Doerflinger, November 1983

SEL-83-007, *Proceedings From the Eighth Annual Software Engineering Workshop*, November 1983

SEL-83-106, *Monitoring Software Development Through Dynamic Variables (Revision 1)*, C. W. Doerflinger, November 1989

SEL-84-001, *Manager's Handbook for Software Development*, W. W. Agresti, F. E. McGarry, D. N. Card, et al., April 1984

SEL-84-003, *Investigation of Specification Measures for the Software Engineering Laboratory (SEL)*, W. W. Agresti, V. E. Church, and F. E. McGarry, December 1984

SEL-84-004, *Proceedings From the Ninth Annual Software Engineering Workshop*, November 1984

SEL-85-001, *A Comparison of Software Verification Techniques*, D. N. Card, R. W. Selby, Jr., F. E. McGarry, et al., April 1985

SEL-85-002, *Ada Training Evaluation and Recommendations From the Gamma Ray Observatory Ada Development Team*, R. Murphy and M. Stark, October 1985

SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985

SEL-85-004, *Evaluations of Software Technologies: Testing, CLEANROOM, and Metrics*, R. W. Selby, Jr., May 1985

SEL-85-005, *Software Verification and Testing*, D. N. Card, C. Antle, and E. Edwards, December 1985

SEL-85-006, *Proceedings From the Tenth Annual Software Engineering Workshop*, December 1985

SEL-86-001, *Programmer's Handbook for Flight Dynamics Software Development*, R. Wood and E. Edwards, March 1986

SEL-86-002, *General Object-Oriented Software Development*, E. Seidewitz and M. Stark, August 1986

SEL-86-003, *Flight Dynamics System Software Development Environment Tutorial*, J. Buell and P. Myers, July 1986

SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986

SEL-86-005, *Measuring Software Design*, D. N. Card, October 1986

SEL-86-006, *Proceedings From the Eleventh Annual Software Engineering Workshop*, December 1986

SEL-87-001, *Product Assurance Policies and Procedures for Flight Dynamics Software Development*, S. Perry et al., March 1987

SEL-87-002, *Ada Style Guide (Version 1.1)*, E. Seidewitz et al., May 1987

SEL-87-003, *Guidelines for Applying the Composite Specification Model (CSM)*, W. W. Agresti, June 1987

SEL-87-004, *Assessing the Ada Design Process and Its Implications: A Case Study*, S. Godfrey, C. Brophy, et al., July 1987

SEL-87-008, *Data Collection Procedures for the Rehosted SEL Database*, G. Heller, October 1987

SEL-87-009, *Collected Software Engineering Papers: Volume V*, S. DeLong, November 1987

SEL-87-010, *Proceedings From the Twelfth Annual Software Engineering Workshop*, December 1987

SEL-88-001, *System Testing of a Production Ada Project: The GRODY Study*, J. Seigle, L. Esker, and Y. Shi, November 1988

SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988

SEL-88-003, *Evolution of Ada Technology in the Flight Dynamics Area: Design Phase Analysis*, K. Quimby and L. Esker, December 1988

SEL-88-004, *Proceedings of the Thirteenth Annual Software Engineering Workshop*, November 1988

SEL-88-005, *Proceedings of the First NASA Ada User's Symposium*, December 1988

SEL-89-002, *Implementation of a Production Ada Project: The GRODY Study*, S. Godfrey and C. Brophy, September 1989

SEL-89-003, *Software Management Environment (SME) Concepts and Architecture*, W. Decker and J. Valett, August 1989

SEL-89-004, *Evolution of Ada Technology in the Flight Dynamics Area: Implementation/Testing Phase Analysis*, K. Quimby, L. Esker, L. Smith, M. Stark, and F. McGarry, November 1989

SEL-89-005, *Lessons Learned in the Transition to Ada From FORTRAN at NASA/Goddard*, C. Brophy, November 1989

SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989

SEL-89-007, *Proceedings of the Fourteenth Annual Software Engineering Workshop*, November 1989

SEL-89-008, *Proceedings of the Second NASA Ada Users' Symposium*, November 1989

SEL-89-101, *Software Engineering Laboratory (SEL) Database Organization and User's Guide (Revision 1)*, M. So, G. Heller, S. Steinberg, K. Pumphrey, and D. Spiegel, February 1990

SEL-90-001, *Database Access Manager for the Software Engineering Laboratory (DAMSEL) User's Guide*, M. Buhler and K. Pumphrey, March 1990

SEL-90-002, *The Cleanroom Case Study in the Software Engineering Laboratory: Project Description and Early Analysis*, S. Green et al., March 1990

SEL-90-003, *A Study of the Portability of an Ada System in the Software Engineering Laboratory (SEL)*, L. O. Jun and S. R. Valett, June 1990

SEL-90-004, *Gamma Ray Observatory Dynamics Simulator in Ada (GRODY) Experiment Summary*, T. McDermott, September 1990

SEL-RELATED LITERATURE

⁴Agresti, W. W., V. E. Church, D. N. Card, and P. L. Lo, "Designing With Ada for Satellite Simulation: A Case Study," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

[2]Agresti, W. W., F. E. McGarry, D. N. Card, et al., "Measuring Software Technology," *Program Transformation and Programming Environments*. New York: Springer-Verlag, 1984

[1]Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," *Proceedings of the Fifth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1981

[7]Basili, V. R., *Maintenance = Reuse-Oriented Software Development*, University of Maryland, Technical Report TR-2244, May 1989

[1]Basili, V. R., "Models and Metrics for Software Management and Engineering," *ASME Advances in Computer Technology*, January 1980, vol. 1

[7]Basili, V. R., *Software Development: A Paradigm for the Future*, University of Maryland, Technical Report TR-2263, June 1989

Basili, V. R., *Tutorial on Models and Metrics for Software Management and Engineering*. New York: IEEE Computer Society Press, 1980 (also designated SEL-80-008)

[3]Basili, V. R., "Quantitative Evaluation of Software Methodology," *Proceedings of the First Pan-Pacific Computer Conference, September 1985*

[1]Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?," *Journal of Systems and Software*, February 1981, vol. 2, no. 1

[1]Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," *Journal of Systems and Software*, February 1981, vol. 2, no. 1

[3]Basili, V. R., and N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," *Proceedings of the International Computer Software and Applications Conference*, October 1985

[4]Basili, V. R., and D. Patnaik, *A Study on Fault Prediction and Reliability Assessment in the SEL Environment*, University of Maryland, Technical Report TR-1699, August 1986

[2]Basili, V. R., and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Communications of the ACM*, January 1984, vol. 27, no. 1

[1]Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," *Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics*, March 1981

Basili, V. R., and J. Ramsey, *Structural Coverage of Functional Testing*, University of Maryland, Technical Report TR-1442, September 1984

[3]Basili, V. R., and C. L. Ramsey, "ARROWSMITH-P—A Prototype Expert System for Software Engineering Management," *Proceedings of the IEEE/MITRE Expert Systems in Government Symposium*, October 1985

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," *Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity, and Cost. New York: IEEE Computer Society Press*, 1979

[5]Basili, V., and H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," *Proceedings of the 9th International Conference on Software Engineering*, March 1987

[5]Basili, V., and H. D. Rombach, "T A M E: Tailoring an Ada Measurement Environment," *Proceedings of the Joint Ada Conference*, March 1987

[5]Basili, V., and H. D. Rombach, "T A M E: Integrating Measurement Into Software Environments," University of Maryland, Technical Report TR-1764, June 1987

[6]Basili, V. R., and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, June 1988

[7]Basili, V. R., and H. D. Rombach, *Towards A Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment*, University of Maryland, Technical Report TR-2158, December 1988

[2]Basili, V. R., R. W. Selby, Jr., and T. Phillips, "Metric Analysis and Data Validation Across FORTRAN Projects," *IEEE Transactions on Software Engineering*, November 1983

[3]Basili, V. R., and R. W. Selby, Jr., "Calculation and Use of an Environments's Characteristic Software Metric Set," *Proceedings of the Eighth International Conference on Software Engineering. New York: IEEE Computer Society Press*, 1985

Basili, V. R., and R. W. Selby, Jr., *Comparing the Effectiveness of Software Testing Strategies*, University of Maryland, Technical Report TR-1501, May 1985

[3]Basili, V. R., and R. W. Selby, Jr., "Four Applications of a Software Data Collection and Analysis Methodology," *Proceedings of the NATO Advanced Study Institute*, August 1985

[4]Basili, V. R., R. W. Selby, Jr., and D. H. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering*, July 1986

[5]Basili, V. and R. Selby, Jr., "Comparing the Effectiveness of Software Testing Strategies," *IEEE Transactions on Software Engineering*, December 1987

[2]Basili, V. R., and D. M. Weiss, *A Methodology for Collecting Valid Software Engineering Data*, University of Maryland, Technical Report TR-1235, December 1982

[3]Basili, V. R., and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, November 1984

[1]Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," *Proceedings of the Fifteenth Annual Conference on Computer Personnel Research*, August 1977

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," *Proceedings of the Software Life Cycle Management Workshop*, September 1977

[1]Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," *Proceedings of the Second Software Life Cycle Management Workshop*, August 1978

[1]Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," *Computers and Structures, August 1978, vol. 10*

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," *Proceedings of the Third International Conference on Software Engineering.* New York: IEEE Computer Society Press, 1978

[5]Brophy, C., W. Agresti, and V. Basili, "Lessons Learned in Use of Ada-Oriented Design Methods," *Proceedings of the Joint Ada Conference*, March 1987

[6]Brophy, C. E., S. Godfrey, W. W. Agresti, and V. R. Basili, "Lessons Learned in the Implementation Phase of a Large Ada Project," *Proceedings of the Washington Ada Technical Conference*, March 1988

[2]Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

[2]Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982

[3]Card, D. N., "A Software Technology Evaluation Program," *Annais do XVIII Congresso Nacional de Informatica*, October 1985

[5]Card, D., and W. Agresti, "Resolving the Software Science Anomaly," *The Journal of Systems and Software*, 1987

[6]Card, D. N., and W. Agresti, "Measuring Software Design Complexity," *The Journal of Systems and Software*, June 1988

Card, D. N., V. E. Church, W. W. Agresti, and Q. L. Jordan, "A Software Engineering View of Flight Dynamics Analysis System," Parts I and II, Computer Sciences Corporation, Technical Memorandum, February 1984

[4]Card, D. N., V. E. Church, and W. W. Agresti, "An Empirical Study of Software Design Practices," *IEEE Transactions on Software Engineering*, February 1986

Card, D. N., Q. L. Jordan, and V. E. Church, "Characteristics of FORTRAN Modules," Computer Sciences Corporation, Technical Memorandum, June 1984

[5]Card, D., F. McGarry, and G. Page, "Evaluating Software Engineering Technologies," *IEEE Transactions on Software Engineering*, July 1987

[3]Card, D. N., G. T. Page, and F. E. McGarry, "Criteria for Software Modularization," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

[1]Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," *Proceedings of the Fifth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1981

[4]Church, V. E., D. N. Card, W. W. Agresti, and Q. L. Jordan, "An Approach for Assessing Software Prototypes," *ACM Software Engineering Notes*, July 1986

[2]Doerflinger, C. W., and V. R. Basili, "Monitoring Software Development Through Dynamic Variables," *Proceedings of the Seventh International Computer Software and Applications Conference*. New York: IEEE Computer Society Press, 1983

[5]Doubleday, D., "ASAP: An Ada Static Source Code Analyzer Program," University of Maryland, Technical Report TR-1895, August 1987 (NOTE: 100 pages long)

[6]Godfrey, S., and C. Brophy, "Experiences in the Implementation of a Large Ada Project," *Proceedings of the 1988 Washington Ada Symposium*, June 1988

Hamilton, M., and S. Zeldin, *A Demonstration of AXES for NAVPAK*, Higher Order Software, Inc., TR-9, September 1977 (also designated SEL-77-005)

Jeffery, D. R., and V. Basili, *Characterizing Resource Data: A Model for Logical Association of Software Data*, University of Maryland, Technical Report TR-1848, May 1987

[6]Jeffery, D. R., and V. R. Basili, "Validating the TAME Resource Data Model," *Proceedings of the Tenth International Conference on Software Engineering*, April 1988

[5]Mark, L., and H. D. Rombach, *A Meta Information Base for Software Engineering*, University of Maryland, Technical Report TR-1765, July 1987

[6]Mark, L., and H. D. Rombach, "Generating Customized Software Engineering Information Bases From Software Process and Product Specifications," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989

[5]McGarry, F., and W. Agresti, "Measuring Ada for Software Development in the Software Engineering Laboratory (SEL)," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

[7]McGarry, F., L. Esker, and K. Quimby, "Evolution of Ada Technology in a Production Software Environment," *Proceedings of the Sixth Washington Ada Symposium (WADAS)*, June 1989

[3]McGarry, F. E., J. Valett, and D. Hall, "Measuring the Impact of Computer Resource Quality on the Software Development Process and Product," *Proceedings of the Hawaiian International Conference on System Sciences*, January 1985

National Aeronautics and Space Administration (NASA), *NASA Software Research Technology Workshop (Proceedings)*, March 1980

[3]Page, G., F. E. McGarry, and D. N. Card, "A Practical Experience With Independent Verification and Validation," *Proceedings of the Eighth International Computer Software and Applications Conference*, November 1984

[5]Ramsey, C., and V. R. Basili, *An Evaluation of Expert Systems for Software Engineering Management*, University of Maryland, Technical Report TR-1708, September 1986

[3]Ramsey, J., and V. R. Basili, "Analyzing the Test Process Using Structural Coverage," *Proceedings of the Eighth International Conference on Software Engineering*. New York: IEEE Computer Society Press, 1985

[5]Rombach, H. D., "A Controlled Experiment on the Impact of Software Structure on Maintainability," *IEEE Transactions on Software Engineering*, March 1987

[6]Rombach, H. D., and V. R. Basili, "Quantitative Assessment of Maintenance: An Industrial Case Study," *Proceedings From the Conference on Software Maintenance*, September 1987

[6]Rombach, H. D., and L. Mark, "Software Process and Product Specifications: A Basis for Generating Customized SE Information Bases," *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, January 1989

[7]Rombach, H. D., and B. T. Ulery, *Establishing a Measurement Based Maintenance Improvement Program: Lessons Learned in the SEL*, University of Maryland, Technical Report TR-2252, May 1989

[5]Seidewitz, E., "General Object-Oriented Software Development: Background and Experience," *Proceedings of the 21st Hawaii International Conference on System Sciences*, January 1988

[6]Seidewitz, E., "General Object-Oriented Software Development with Ada: A Life Cycle Approach," *Proceedings of the CASE Technology Conference*, April 1988

[6]Seidewitz, E., "Object-Oriented Programming in Smalltalk and Ada," *Proceedings of the 1987 Conference on Object-Oriented Programming Systems, Languages, and Applications*, October 1987

[4]Seidewitz, E., and M. Stark, "Towards a General Object-Oriented Software Development Methodology," *Proceedings of the First International Symposium on Ada for the NASA Space Station*, June 1986

[7]Stark, M. E. and E. W. Booth, "Using Ada to Maximize Verbatim Software Reuse," *Proceedings of TRI-Ada 1989*, October 1989

Stark, M., and E. Seidewitz, "Towards a General Object-Oriented Ada Lifecycle," *Proceedings of the Joint Ada Conference*, March 1987

[7]Sunazuka. T., and V. R. Basili, *Integrating Automated Support for a Software Management Cycle Into the TAME System*, University of Maryland, Technical Report TR-2289, July 1989

Turner, C., and G. Caron, *A Comparison of RADC and NASA/SEL Software Development Data, Data and Analysis Center for Software*, Special Publication, May 1981

Turner, C., G. Caron, and G. Brement, *NASA/SEL Data Compendium, Data and Analysis Center for Software*, Special Publication, April 1981

[5]Valett, J., and F. McGarry, "A Summary of Software Measurement Experiences in the Software Engineering Laboratory," *Proceedings of the 21st Annual Hawaii International Conference on System Sciences*, January 1988

[3]Weiss, D. M., and V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data From the Software Engineering Laboratory," *IEEE Transactions on Software Engineering*, February 1985

[5]Wu, L., V. Basili, and K. Reed, "A Structure Coverage Tool for Ada Software Systems," *Proceedings of the Joint Ada Conference*, March 1987

[1]Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," *Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York:* IEEE Computer Society Press, 1979

[2]Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," *Empirical Foundations for Computer and Information Science* (proceedings), November 1982

[6]Zelkowitz, M. V., "The Effectiveness of Software Prototyping: A Case Study," *Proceedings of the 26th Annual Technical Symposium of the Washington, D. C., Chapter of the ACM,* June 1987

[6]Zelkowitz, M. V., "Resource Utilization During Software Development," *Journal of Systems and Software*, 1988

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," *Proceedings of the Software Life Cycle Management Workshop*, September 1977

NOTES:

[1]This article also appears in SEL-82-004, *Collected Software Engineering Papers: Volume I*, July 1982.

[2]This article also appears in SEL-83-003, *Collected Software Engineering Papers: Volume II*, November 1983.

[3]This article also appears in SEL-85-003, *Collected Software Engineering Papers: Volume III*, November 1985.

[4]This article also appears in SEL-86-004, *Collected Software Engineering Papers: Volume IV*, November 1986.

[5]This article also appears in SEL-87-009, *Collected Software Engineering Papers: Volume V*, November 1987.

[6]This article also appears in SEL-88-002, *Collected Software Engineering Papers: Volume VI*, November 1988.

[7]This article also appears in SEL-89-006, *Collected Software Engineering Papers: Volume VII*, November 1989.